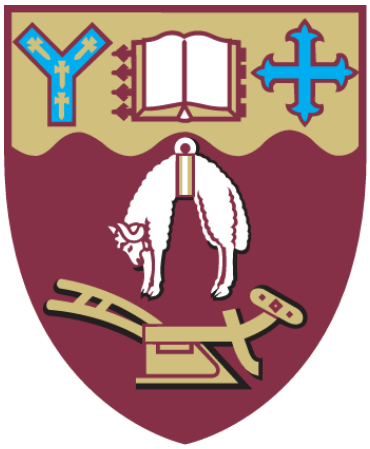
Tree Position Detection for Autonomous UAV Navigation



Peiwen Luo

Department of Computer Science and Software Engineering

University of Canterbury

This dissertation is submitted for the degree of

Master of Science

College of Science July 2017

ACKNOWLEDGEMENTS

Firstly, I would like to acknowledge my primary supervisor Richard Green for all his advice and guidance throughout my project. I am very fortunate and grateful to have such a great supervisor who has given the best advice I have received for a long time. The knowledge and skills I have gained from working with him will be invaluable in my future career.

Secondly, I would also like to thank our UAV pruning project team leader Matthew Edwards for his knowledge and suggestions.

In addition, I would like to thank all the UAV pruning project team members as well as the chats and laughs we shared together.

A special thanks to Ori Ganoni for all his technical support in the computer graphics area and the scene modeling of the simulator in the project.

Furthermore, I would also like to thank all the university staff involved for their assistance.

I wish to express my appreciation to my family and friends for all the support they have given me throughout my whole life.

I would like to acknowledge my appreciation of being a recipient of the University of Canterbury Masters Scholarship. This support has been of great assistance in the completion of this research.

Finally, I would like to thank the whole team who participated in this study. Without the support of key people in the team, there would be no dissertation. I hope this research will provide some help for this pruning tree project.

ABSTRACT

This research proposes tree detection and location methods using RGB-D data. The first proposed approach uses a simultaneous localization and mapping (SLAM) algorithm based on RGB-D image data to build a dense point cloud map. Through reducing the dimension of the point cloud map from 3D to 2D via a slicing method, and the Euler clustering algorithm, the system locates the approximate position of the trees around the camera within a certain range. Finally, when an approximate tree position is within the range of the depth camera, the system uses a merged depth map to detect and adjust the exact location of this tree in real time.

The second approach proposes an autonomous navigation algorithm to control an unmanned aerial vehicle (UAV) using a novel tree detection and navigation process. The navigation system controls the UAV to take off, rotate 360^{o} to scan the surrounding scenes and navigate to the nearest tree by providing instructions from ROS to the PX4 flight controller. Kalman filtering improves the robustness and fault tolerance capability of the navigation by adjusting the relative position of the detected tree with respect to the camera.

From our 20 experiments, the proposed method has 100% correct tree detection rate in single tree scenes, 90.9% correct tree detection rate in multiple tree scenes with trees close together and 2.5 times faster calculation speed than prior research which only achieved an accuracy of 66%-89%.

CONTENTS

1 INTRODUCTION 1
1.1 OBJECTIVES OF THE THESIS 5
1.2 STRUCTURE OF THE THESIS 5
1.3 CONTRIBUTION OF THE THESIS 6
2 BACKGROUND 7
2.1 ROBOT OPERATING SYSTEM 7
2.2 UAV FLIGHT CONTROLLER AND SIMULATOR 10
2.3 VISUAL SLAM 11
2.3.1 Visual Odometry 15
2.3.2 Optimization and Loop Closure Detection 17
2.3.3 Mapping 18
2.4 OBJECT DETECTION AND LOCALISATION 19
2.4.1 Ground Removal 20
2.4.2 Tree Detection 21
2.5 NAVIGATION 22
2.6 SUMMARY 23
3 METHODOLOGY 25
3.1 DATA ACQUISITION 26
3.1.1 Hardware Environment 27
3.1.2 Software Environment 30
3.1.3 Depth Map Denoise and Hole Filling 31
3.2 SIMULTANEOUS LOCALISATION AND MAPPING 35
3.2.1 Camera Calibration 36
3.2.2 Motion Estimation 36
3.2.3 3D Point Cloud Reconstruction 41
3.2.4 Loop Closure Detection 45
3.3 TREE DETECTION 46
3.3.1 Tree Detection in 3D Point Cloud 46

3.3.2 Tree Detection in 2D Image *52*
3.4 NAVIGATION IN ROS 55
3.4.1 UAV Movement Process *56*
3.4.2 Handheld Camera Simulation *57*
3.5 CHAPTER SUMMARY 59
4 RESULT AND DISCUSSION 60
4.1 TREE DETECTION 60
4.1.1 Depth Threshold *60*
4.1.2 Data Filter *61*
4.1.3 Ground Removal *62*
4.1.4 Point Cloud Detection *63*
4.1.5 Depth Image Detection *70*
4.2 NAVIGATION 74
4.2.1 Handheld Camera Experiment *74*
5 CONCLUSION AND FUTURE WORK 76
5.1 FUTURE RESEARCH 77
5.1.1 Data Acquisition *77*
5.1.2 Point Cloud Mapping *78*
5.1.3 Complete Navigation System *78*
5.1.4 System *78*
6 REFERENCES 79
7 APPENDICES 88
APPENDIX A CAMERA CALIBRATION 89
APPENDIX B STRUCTURE FROM MOTION 91

LIST OF TABLES

TABLE 2.1 THE AVERAGE TIMES OF SAME FEATURES POINT NUMBER OF 24 640X480 IMAGES. 17

TABLE 2.2 THE TREE DETECTION RATE FROM THREE DIFFERENT METHODS, TIFFS, FUSION, LAYER STACKING AND HORIZONTAL DIRECTED NORMAL VECTORS DETECTION..... 22

TABLE 3.1 COMPARISON OF THESE THREE COMMON DEPTH DETECTION TECHNOLOGIES, STRUCTURED LIGHT, TIME OF FLIGHT AND STEREO CAMERA. 28

TABLE 4.1 THE COMPARISON OF DENSITY AND SLICING THICKNESS TO COMPARE DIFFERENT VOXEL LEAF SIZES. UNITS ARE IN METERS EXCEPT RUNNING TIME. 63

TABLE 4.2 THE TABLE OF THE RELATIONSHIP BETWEEN TRUNK RADIUS WITH THE NUMBER OF POINTS IN EACH SLICED LAYER OF THE TRUNK. 65

TABLE 4.3 THE EUCLIDEAN CLUSTERING ALGORITHM PARAMETERS COMPARISON FOR THE SLICING 2D POINTS..... 65

TABLE 4.4 THE RESULT OF 3D POINT CLOUD TREE DETECTION. 69

TABLE 4.5 THE COMPARISON OF AVERAGE TIME COST AND TRUE POSITIVE RATE OF POINT CLOUD SLICING AND CYLINDER FITTING TREE DETECTION METHOD THROUGH POINT CLOUD AND DEPTH IMAGE. 74

LIST OF FIGURES

FIGURE 1.1 THE LEFT IMAGE IS THE KNOT FROM A PINE BRANCH PRUNED BY FORESTRY WORKERS. THE RIGHT IMAGE IS THE KNOT FREE TIMBER. 1

FIGURE 1.2 THE LEFT IMAGE SHOWS A PRUNED PINE TREE OF FIVE TO SIX METERS HEIGHT. THE RIGHT IMAGE SHOWS AN EIGHT METER PINE TREE BEING PRUNED. IMAGES PROVIDED BY [NEW ZEALAND FARM FORESTRY ASSOCIATION]. 2

FIGURE 2.1 THE SIMPLE DIAGRAM SHOWS THE ROS MESSAGE COMMUNICATION METHOD BETWEEN EACH NODE THROUGH TOPIC AND SERVICE. THE LEFT SIDE NODES WILL BE RESPONSIBLE FOR READING DATA FROM THE HARDWARE DRIVER. THE DATA WILL BE PACKAGED WITH THE MESSAGE, AND ROS WILL IDENTIFY AND DISTRIBUTE THE MESSAGE TO OTHER NODES ON RIGHT SIDE. THE IMAGE COMES FROM [MAZZARI, 2016]. 9

FIGURE 2.2 THE ILLUSTRATION OF ROS NODE COMMUNICATION EXAMPLE BETWEEN A CAMERA WITH IMAGE PROCESSING NODES REGISTERED WITH THE ROS MASTER. THE IMAGE COMES FROM [CLEARPATH ROBOTICS, 2014]. 10

FIGURE 2.3 THE GENERAL FLOWCHART OF THE COMPLETE PROCESS OF V-SLAM. 14

FIGURE 2.4 THE TOP-LEFT IMAGE SHOWS THE 2D GRID-BASE MAP. THE TOP-RIGHT SHOWS THE 2D TOPOLOGICAL MAP, IMAGE FROM [THRUN ET AL. 1998]. THE BOTTOM LEFT IMAGE SHOWS THE 3D OCTO-MAP, AND THE BOTTOM RIGHT IMAGE SHOWS THE 3D DENSE POINT CLOUD MAP. 19

FIGURE 3.1 THE FLOW CHART OF THE TREE DETECTION AND NAVIGATION ALGORITHM. RGB (GRAY-SCALE) AND DEPTH (REGISTERED) SEQUENCES ARE REQUIRED TO ESTIMATE CAMERA POSE AND GENERATE POINT CLOUD BY OUR MODIFIED ORBSLAM2. TREES ARE DETECTED AFTER THE SURROUNDING POINT CLOUD IS GENERATED WITH LOOP CLOSURE AND FULL BUNDLE ADJUSTMENT. RELATED APPROXIMATE POSITIONS BETWEEN TREES AND CAMERA CAN BE ESTIMATED. THE SPECIFIC TREE'S LOCATION CAN ALSO BE ADJUSTED TO THE APPROXIMATE POSITION OF TREES USING THE CURRENT RGB-D FRAME IN REAL-TIME WHEN A DETECTED TREE (FROM POINT CLOUD) IS WITHIN THE FIELD OF VIEW OF THE CAMERA. FINALLY, THE SYSTEM SENDS A MESSAGE AS A ROS TOPIC TO CONTROL A UAV TOWARD A TARGET TREE. 26

FIGURE 3.2 THE TOP IMAGES SHOW A VELODYNE HDL-64E LIDAR SENSOR AND A BUMBLEBEE2 STEREO CAMERA. THE BOTTOM IMAGE SHOWS A KINECT V1 AND A KINECT V2, AND AN INTEL REALSENSE R200. 27

FIGURE 3.3 THE FRONT VIEW OF INTEL® REALSENSE™ CAMERA R200 WITHOUT REINFORCEMENT FRAME TO EXPOSE COMPONENTS. THE IMAGE COMES FROM [R200 OFFICIAL DATASHEET]. 30

FIGURE 3.4 GAZEBO 7 FOUR-AXIS UAV SIMULATION. 31

FIGURE 3.5 R200 CAMERA DEPTH IMAGES. (A) THE IMAGE IS RAW DEPTH IMAGE IN THE INDOOR ENVIRONMENT. (B) THIS IMAGE IS THE REGISTERED DEPTH IMAGE. (C) THE IMAGE IS THE PRE-PROCESSED REGISTERED DEPTH IMAGE. 31

FIGURE 3.6 THE WORKFLOW OF VISUAL SLAM. 36

FIGURE 3.7 THE POINT CLOUD IMAGE ONLY CONTAINS GROUND AND TREE WITHOUT THE CANOPY 41

FIGURE 3.8 ORB-SLAM2 SPARSE POINT CLOUD MAP VIEWER BASED ON PANGOLIN. THE BLUE LINE (OVERLAPPING BLUE SQUARES) ARE THE TRAJECTORY OF THE CAMERA. THE RED POINTS ARE CURRENT FEATURE POINTS IN THE CAMERA. 42

FIGURE 3.9 THE PROJECT METHOD SCHEMATIC DIAGRAM. SIX 3D POINTS (RED) ARE PROJECTED TO THE SLICE PLANE $S0$ AS THREE INTERSECTION POINTS (GREEN). 48

FIGURE 3.10 THE FIGURE SHOWS THE 2D SLICED POINTS CLUSTERING FOR ONE OF THE SLICED POINT CLOUDS. THE DIFFERENT COLOR POINTS MEAN THE DIFFERENT CLUSTERS. 49

FIGURE 3.11 TREE DETECTION VISUALIZATION IN THE 3D POINT CLOUD. 51

IN ADDITION, LOCATION OF A TREE CAN BE DETERMINED IN A 2D IMAGE AS SHOWN AS A RED RECTANGLE IN BELOW FIGURE 3.12 AND THE RADIUS OF THE TREE CAN ALSO BE DISPLAYED IN THE 2D IMAGE. MOREOVER, THE PROJECTED TREE RADIUS CAN ALSO BE CHANGED WITH THE MODIFICATION OF THE DISTANCE BETWEEN THE TREE AND THE CAMERA. HOWEVER, THERE IS A CERTAIN DEVIATION BETWEEN THE PROJECTED TREE POSITION AND THE GROUND TRUTH TREE POSITION IN THE 2D IMAGE, AS SHOWN IN FIGURE 3.13 THIS DEVIATION HAS HAD A SIGNIFICANT IMPACT ON UAV NAVIGATION. THEREFORE, THIS SECTION WILL DESCRIBE A METHOD TO ADJUST THE POSITION OF THE TREE USING THE DEPTH IMAGE. 52

FIGURE 3.13 AN ESTIMATED TREE POSITION IN 2D RGB-D CAMERA IMAGE FROM A 3D TREE LOCATION. THE GROUND TRUTH IS THE IMAGE OF THE TREE, AND THE MAPPED TREE IS SHOWN AS A RED RECTANGLE. THE POSITION OF THE TREE IS MEASURED WITHOUT LOOP CLOSURE DETECTION. 52

FIGURE 3.14 THE SETUP OF UAV FLIGHT CONTROLLER SIMULATOR GAZEBO 7 BASED ON ROS. 56

FIGURE 3.15 THE HANDHELD CAMERA TREE DETECTION NAVIGATION SIMULATION TRAJECTORY. 57

FIGURE 3.16 TOP-DOWN VIEW: THE ROTATION CONTROL OF THE CAMERA WHEN THE CLOSEST TREE IS NOT IN THE FIELD OF VIEW. 58

FIGURE 4.1 (A) THE RAW DEPTH REGISTERS AN IMAGE FROM R200 VIA ROS LINUX DRIVER. (B) THE DEPTH IMAGE IN THE RANGE 0.5 METERS TO SIX METERS. 60

FIGURE 4.2 (A) THE RAW POINT CLOUD GENERATED VIA THE RAW DEPTH IMAGE WITHOUT RANGE THRESHOLD INCLUDES 1,856,481 POINTS. (B) THE RAW POINT CLOUD AFTER THE 0.5M TO 6M THRESHOLD CUT INCLUDES 1,073,888 POINTS. 61

FIGURE 4.3 (TOP) THE POINT CLOUD MAP THROUGH THE DOWNSAMPLING FILTER INCLUDES 238,432 POINTS. (BOTTOM) THE POINT CLOUD MAP THROUGH THE STATISTICAL OUTLIER REMOVAL FILTER INCLUDES 216,219 POINTS. 62

FIGURE 4.4 (A) THE GROUND POINTS EXTRACTED FROM OUTLIER REMOVED CLOUD MAP. (B) THE TREE AND OTHER OBJECTS POINT ABOVE THE GROUND AFTER THE GROUND REMOVAL ALGORITHM. 63

FIGURE 4.5 THE SLICED POINT CLOUD MAP. IN THIS CASE, THE DOWN SAMPLING VOXEL LEAF SIZE IS 0.01. THE SLICING THICKNESS IS 0.0281 METERS. A TREE TRUNK IS SLICED INTO NINE LAYERS. EACH LAYER HAS A SIMILAR CENTER AND RADIUS. 64

FIGURE 4.6 THE TWO SLICED 2D CIRCLES FOR ONE LAYER OF AN APPROXIMATELY 10CM RADIUS TREE TRUNK WITH 547 POINTS AND 20CM RADIUS TREE TRUNK WITH 1109. 65

FIGURE 4.7 CLUSTERING RESULTS OF THE 2D EUCLIDEAN CLUSTERING ALGORITHM IN DIFFERENT EUCLIDEAN DISTANCE AND MINIMUM/MAXIMUM CLUSTER SIZE. 66

FIGURE 4.8 THE RESULT OF CIRCLE FITTING. (A) EXTRACTED POINTS FROM CLUSTERED POINTS VIA CONVEX HULL ALGORITHM [SUSAN ET AL. 2017]. (B) THE VISUALIZATION

OF FITTED CIRCLE VIA THE LEVENBERG-MARQUARDT CIRCLE FITTING ALGORITHM [CHERNOV IN 2010]. 67

FIGURE 4.9 THE FITTED CIRCLES IN THE 3D POINT CLOUD MAP. THE TWO RED AREAS REPRESENT THE OBJECTS. THERE ARE TWO TREES IN THESE TWO SPACES. 68

FIGURE 4.10 THE VISUALIZATION POINT CLOUD MAP OF THE SUITABLE TREE DETECTION. THE RED CUBE PRESENTS THE DETECTED TREE POSITION. IN THE RIGHT AREA, A BIG TREE WITH A 40CM RADIUS IS IGNORED BY THE ALGORITHM. IN THE LEFT AREA, THERE IS A TREE WHERE THE TRUNK OF THE TREE IS BEYOND IS THE EFFECTIVE DISTANCE OF THE CAMERA. ONLY THE CANOPY OF THE TREE FORMS THE POINT CLOUD. 68

FIGURE 4.11 THE CASE OF FALSE POSITIVE TREE DETECTION FOR THE 3D TREE DETECTION ALGORITHM BASED ON POINT CLOUD. 69

FIGURE 4.12 (A) THE REGISTERED DEPTH IMAGE. (B) THE DEPTH IMAGE THROUGH MORPHOLOGICAL DILATION. (C) THE CONTOUR OF THE TREE. (D) THE FINAL TREE DETECTION RESULT. 70

FIGURE 4.13 THE COMPARISON OF RAW TREE POSITION AND KALMAN FILTERED DATA IN THE X-AXIS OF A 2D IMAGE. 71

FIGURE 4.14 THE COMPARISON OF RAW TREE POSITION AND KALMAN FILTERED DATA IN THE X-AXIS OF 2D DEPTH IMAGE DETECTION RESULTS. 72

FIGURE 4.15 THE COMPARISON OF POINT CLOUD TREE DETECTION RESULTS IN PURPLE LINE AND DEPTH IMAGE TREE DETECTION RESULTS IN YELLOW LINE. 72

FIGURE 4.16 THE THREE BASIC NAVIGATION STATES DURING THE HANDHELD CAMERA EXPERIMENT, INCLUDE (A) TURN CAMERA LEFT, (B) TURN CAMERA RIGHT, AND (C) GO AHEAD PREPARATION. THE RED ARROW INDICATES THE DIRECTION IN WHICH THE CAMERA NEEDS TO BE ROTATED. THE GREEN RECTANGLE SHOWS THE FINAL DETECTED TREE POSITION. 74

FIGURE 4.17 THE HANDHELD CAMERA NAVIGATION IMAGE SHOWS FROM TOWARD STATES (A) TO DOCKED STATES (B). THE GREEN RECTANGLE SHOWS THE DETECTED TREE POSITION ON THE SCREEN, THE RED ARROW SHOWS GO AHEAD AT TOWARD STATES, AND THE GREEN BORDER AROUND THE DISPLAY SHOWS THE DOCKED STATES. 75

List of Appendices

Appendix A Camera Calibration 89

Appendix B Structure from Motion 91

1 INTRODUCTION

As mentioned by［NZ Ministry of Agriculture and Forestry, 2010], since 1859 people have introduced Pinus Radiata through the United States to New Zealand, where today it has become the most widely-used timber. There are 89% plantation forests of this tree. New Zealand is one of the high-quality timber suppliers in the world. As also mentioned by [Somerville, 1991] and [New Zealand Farm Forestry Association], most of New Zealand's pines are pruned when they are young. This pruning process limits the knots of the pine tree to a small area near the wooden heart to produce the knot-free timber as shown in **Figure 1.1**.

Figure 1.1 The left image is the knot from a pine branch pruned by forestry workers. The right image is the knot free timber.

New Zealand pine trees need to be trimmed three times in the first decade. When these trees grow to five to six meters high, they need their first pruning. Then, before these trees grow to eight meters high, they may be pruned again twice. To ensure pine trees maintain

an efficient photosynthesis, only three to four meters of the canopy needs to be retained as shown in **Figure 1.2**.

Figure 1.2 The left image shows a pruned pine tree of five to six meters height. The right image shows an eight meter pine tree being pruned. Images provided by [New Zealand Farm Forestry Association].

In almost all NZ pine forests, pruning branches is still done manually. [Parker et al. 2004] mentioned in the New Zealand Forest Industry Accident Reporting Scheme, that there were four fatal injuries and 119 hours lost time injuries by round wood removal. Such unsafe factors lead to delays in the production chain, causing more losses for the stakeholders. Safety issues are compounded by pruning costs and shortages of skilled pruners, leading to the motivation of automating pruning forests – where the first step is robustly locating trees.

Simultaneous Localisation and Mapping (SLAM) as mentioned by [Thrun et al. 1998], is used in this research to solve the tree detection problem using the following five steps:

- Data Acquisition – various sensors integrated with ROS.
- Odometry – visual odometry.
- Optimization – filter or graph-base optimization.
- Mapping – 2D/3D environment reconstruction.

- Loop Closure Detection – further optimize trajectories and maps.

The data acquisition issue is a prerequisite for Visual SLAM (V-SLAM).

In order to measure the specific radius of the tree, we use a 3D camera to ensure invariance of the object scale. Through the comparison of several different depth camera characteristics such as LiDAR, structured light camera, time-of-flight camera and a stereo camera, we decided to use an active infrared stereo camera (Intel® RealSense™ Camera R200) as an optical sensor in this research. Although the depth image quality of the R200 camera is not as accurate as others, it is enough for tree detection and the SLAM. In addition, we chose the R200 camera because of its size, weight (UAV load considerations) cost, relatively good driver and high compatibility with the robot operation system (ROS) framework [Quigley et al. 2009].

In this research, the current mainstream open source robot operating system, ROS framework, helps improve efficiency of robot software development. As also mentioned by [Jason in 2014], ROS is a robot-oriented open source meta-operating system that provides several functions similar to the traditional operating system as well as tools and libraries for obtaining, compiling and editing code. Moreover, it can run the program between multiple computers to achieve distributed computing. Currently, most existing SLAM algorithms support ROS, including [Engel et al. in 2014] LSD-SLAM, [Forster et al. 2014] SVO-SLAM, [Mur-Artal et al. 2015] ORB-SLAM, [Endres et al. 2014] RGB-D SLAM and [Labbe et al. 2014] RTAB-Map. Through the comparison of these above algorithms, we selected [Mur-Artal et al. 2015] ORB-SLAM2 as the camera localization and 3D point cloud reconstruction tool in the research.

In the research, we use the R200 camera as the stereo 3D visual odometry data collector. As mentioned by [Scaramuzza et al. 2011], the main method of visual odometry is based on the method of feature point detection, where the feature point method converts a motion estimation of the image into a motion estimation between the two sets of points. Therefore, there are two main problems with the method:

- How do we match the image feature points?
- How do we calculate the camera's motion based on known feature points?

For the first question, based on the comparison of three common feature detection algorithms, ORB, SURF, and SIFT, from [Rublee et al. 2011], we decided to use the ORB

features as the feature point matching algorithm. For the second question, we applied ORBSLAM2 [Mur-Artal et al. 2015] to solve the perspective-3-point-positioning (P3P) problem for the stereo camera to estimate the camera pose.

After the camera pose estimation to solve the P3P problem, an approximate motion can be estimated. However, this method is less tolerant of noise. When there is an error match or a significant deviation in the pixel coordinates, the solution is unreliable. [Mur-Artal et al. 2015] applies a local bundle adjustment and global bundle adjustment to optimize the camera movement trajectory. Thus, it can find the local and global optimal solution. Furthermore, the ORBSLAM2 also uses a simple loop closure detection to improve the optimization through retrieving a binary dictionary. Finally, through the optimized camera trajectory, a point cloud map can be generated via ICP algorithm.

In the research, we achieve tree detection through a reconstructed point cloud map and depth images. The reconstructed point cloud map from the ORBSLAM2 is only a set of scattered points. The 3D point cloud needs to be reduced to clearly segmented objects to determine the number and type of objects in the point cloud map. In this research, we proposed a point cloud slicing method to segment the ground and trees in the point cloud map. This approach reduces the 3D point cloud to multi-layer 2D points map to achieve a relatively fast and accurate detection method. Through the tree detection algorithm, the relative positions of the camera and the sizes of trees are calculated. According to the pose matrix of the camera, and the position and size of trees, these trees can then be approximately projected into the 2D image of the camera.

However, these approximate projected tree positions in the 2D image have some small errors caused by noise or cumulative error of the V-SLAM. To solve the problem, we proposed a tree detection method based on depth images to optimize approximate projected tree positions, where this approach has a good performance in a single tree scene. In 10 single tree scene experiment sets, this method achieved 100% tree position optimization, with a lower performance when there are two trees in an image at the same time.

The process of the navigation has five steps: take-off, 360^{o} rotation/scanning, rotate further to face a target (closest) tree, fly to the tree, and landing. During this process, there are also only four UAV states being used: hovering, turn left, turn right, go ahead and

landing. In the research, we test these primary movement states on a simulator as the preparation for real UAV.

1.1 Objectives of the Thesis

The first task of a larger UAV autonomous navigation pine tree pruning project is to find the locations of the trees. The main goal of this thesis is to enable a UAV to locate trees in an unknown environment using a depth camera with the following objectives:

- To compare four different types of depth cameras for use in forest environments.
- To use the ORBSLAM2 with R200 camera to reconstruct a dense 3D point cloud map.
- To implement a proposed slicing tree detection algorithm based on point clouds.
- To apply a proposed tree detection algorithm based on depth images to optimize the performance of the tree detection algorithm based on point clouds.
- To implement a navigation process from take off to docking with a target tree.

1.2 Structure of the Thesis

This thesis presents a novel approach to detecting trees in the forest with a RGB-D camera and is structured as five chapters.

Chapter II "Background and Related Studies" introduces the basic concepts, advantages and disadvantages of ROS (robot operating system), several UAV flight controllers and simulators. It also introduces some recent related studies in visual SLAM and some object detection and localization methods. Finally, the chapter briefly introduces some background research for autonomous navigation methods, including path planning and obstacle avoidance.

Chapter III "Methodology" details the steps of the tree detection method proposed in this paper and then describes the principle and use of a flight control system simulator. Finally, we describe evaluating the results of the proposed navigation methodology.

Chapter IV "Results and Discussions" provides the tree detection results in correction rate (true positive rate) and the number of false positive cases for both tree detection methods based on the depth image and point cloud. Moreover, the chapter also discusses the simulation results of navigation algorithm.

Chapter V "Conclusion and Future Work" summarizes the experiment results, analyses the problems of the proposed detection and navigation methods, and suggests improvements. Finally, the future research directions of computer vision recognition systems and autonomous navigation systems are discussed.

1.3 Contribution of the Thesis

We reconstructed a dense point cloud map and proposed a novel tree detection algorithm combined with the point cloud map and depth images based on the existing SLAM algorithm, ORBSLAM2. Compared with a prior 3D cylinder model fitting method, this proposed slicing tree detection method based on 3D point cloud has higher accuracy (81.3% to 93.8%) and faster calculation speed (approximate 2.5 times). The depth image based detection algorithm enabled the tree detection system to have more accurate measurement results, including tree distance between the camera and the center of tree and tree radius, in real time. The depth image detection algorithm requires only approximate coordinates of a tree, without the point cloud maps and the specific coordinates of each point.

2 BACKGROUND

This chapter describes the background and some related works about the visual SLAM, object detection, and autonomous navigation algorithms. The following sections also introduce related algorithms in our proposed methodology. Sections 2.1 and 2.2 illustrate some recent research about robot operation systems, UAV flight controllers, and simulators. Section 2.3 introduces related research into visual odometry and visual SLAM in recent years. Then section 2.4 mentions ground removal algorithms and tree detection algorithms using point cloud process. Finally, in section 2.5, some related studies on autonomous navigation are discussed.

2.1 Robot Operating System

Many core technologies of robot research and development are supported by the robot operating system (ROS), which is an open-source meta-operating system developed by [Quigley et al. 2009]. It was created by a collaboration between the project of the Artificial Intelligence Laboratory at Stanford University and a personal robot project from Willow Garage [Quigley et al. 2009]. In both academic and industry robotics research, most relevant algorithms are supported with a ROS version of their source code. For example, many industry robot research teams, such as Google, Microsoft, Boston power, I-Robot, Softbank, Omron and other enterprises, use ROS as the main choice of research and development environment. As mentioned before by [Gerkey, in 2015], more than 150 million US dollars was invested in ROS in recent years. Also, [Greenwald, in 2013] mentioned that since ROS released its 1.0 version, it has become the de facto standard for robot software environment.

[Garage, 2012] also mentioned that ROS is a middleware for robot development based on the Ubuntu Linux system. It provides tools and libraries for acquiring, building, writing and running multi-machine integration programs. It also provides many functions, including inter-program messaging, sharing of functions, program release package management, underlying driver management and hardware abstraction description. In addition, ROS can significantly improve the code effectiveness and reusability in the field of robotics development. In general, ROS has five main features.

- Point-to-point design.

ROS is a distributed processing framework formed by ROS nodes that communicate with each other. ROS is also a processing architecture that uses the ROS communication module to achieve a loosely coupled network connection for point-to-point communication between ROS nodes.

- Multi-language support.

ROS is a multi-language support framework. This feature is reflected mainly in the message communication layer. The XML-RPC mechanism implements the end-to-end connection and configuration. XML-RPC contains many reasonable descriptions of the main programming language.

- Streamlining and Integration.

The system established by ROS is modular; the code in each module is compiled separately. The CMake tool can make it easy to achieve the concept of streamlining, where ROS encapsulates the complex code in the library.

- Toolkit.

In order to manage the complex ROS software framework, a large number of small tools are used to compile and run a variety of ROS tasks, such as organizing the structure of the source code, capturing and setting configuration parameters, visualizing end-to-end topology connections, measuring bandwidth usage, rendering of information data, automatic generation of documents, etc.

- Free and Open Source.

All the ROS source code is publicly available.

One of the biggest contributions of ROS is the development of a unified interface standard for robot development, especially ROS message communication. As [Mazzari, in 2016] mentioned, ROS is a distributed multiprocess framework based on message-passing communication. Because ROS relies on a message mechanism, we can more easily structure software into modules according to their function. Each module encompasses responsibility for reading and distributing messages, where modules communicate through the messages only and these messages include topic and service. The message communication mechanism is shown in **Figure 2.1**.

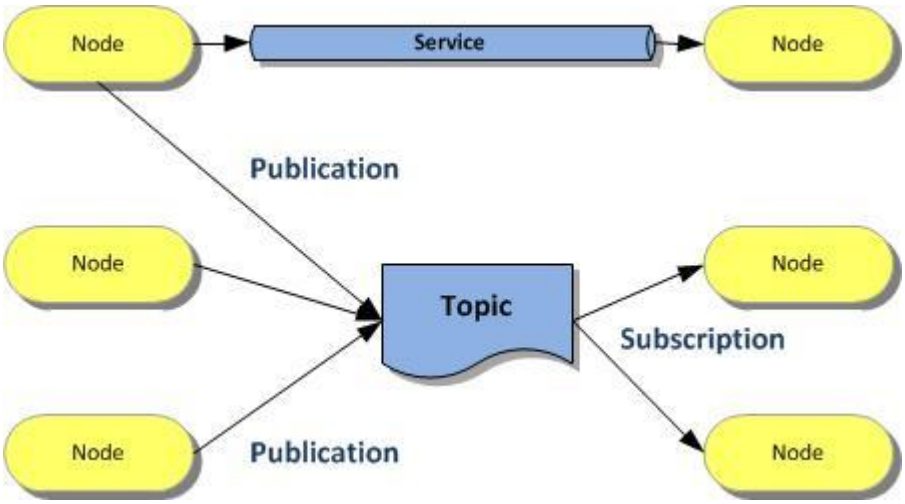


Figure 2.1 The simple diagram shows the ROS message communication method between each node through topic and service. The left side nodes will be responsible for reading data from the hardware driver. The data will be packaged with the message, and ROS will identify and distribute the message to other nodes on right side. The image comes from [Mazzari, 2016].

In this research, the ROS system is used with an RGB-D camera. As mentioned by [Clearpath Robotics, 2014], they use a camera ROS node for communication with the camera and an image processing ROS node to process the visual data. In the beginning, all nodes are registered on the ROS master. The master can be considered as a query table. Each node can query which node need it should message via the master. The node registers a service or topic on the ROS Master. The image processing node accepts the service or topic as shown in **Figure 2.2**.

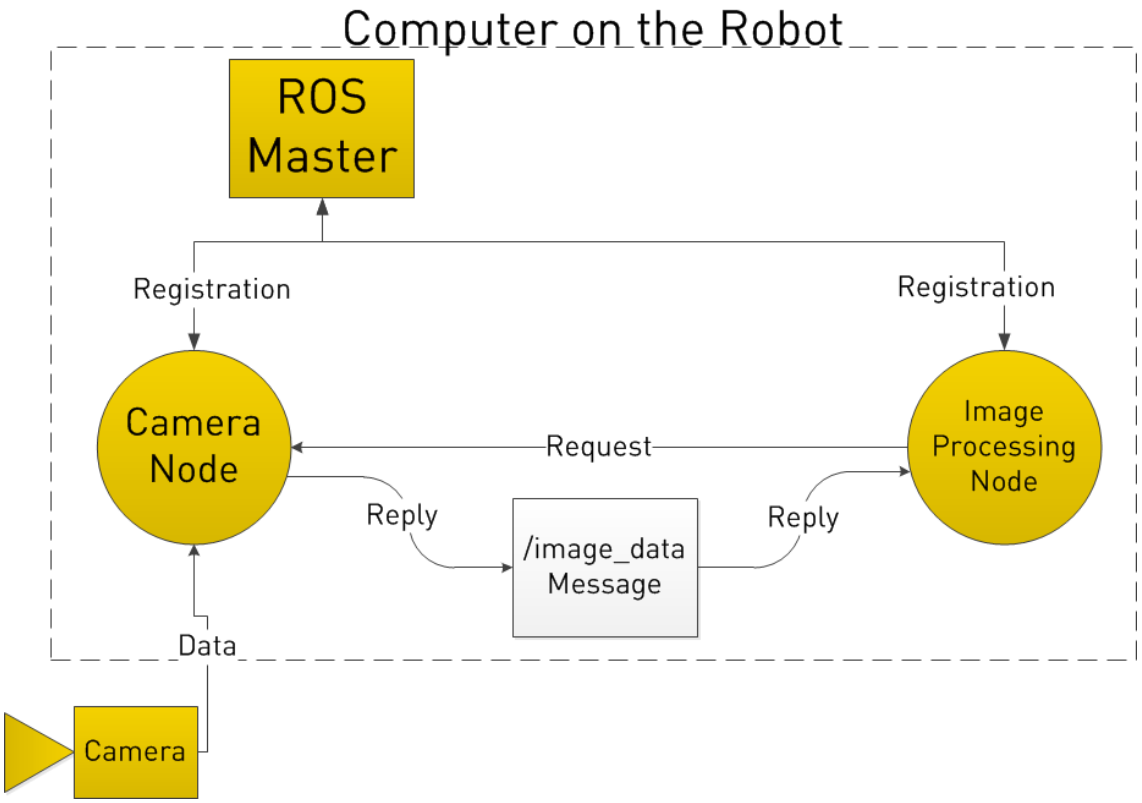


Figure 2.2 The illustration of ROS node communication example between a camera with image processing nodes registered with the ROS master. The image comes from [Clearpath Robotics, 2014].

2.2 UAV Flight Controller and Simulator

As mentioned by [Qi, in 2009], from the perspective of the development process, current flight controllers can be divided into two categories.

- Open source flight controller.

Open source [Perens et al. 2014] flight controllers include firmware in hardware and the ground station software.

- Independent R&D flight controller.

Although a few UAV companies, such as DJI and Ascending, provide some interface to their flight controller, the developer can only develop some high-level functions based on the system, but they cannot optimize these features through the underlying system.

The open source flight controller, PX4, is used in the research.

One of the early flight controllers is based on Arduino and developed by [Mellis et al. in 2005]. Two popular open source flight controllers, WMC and ArduPilot Mega (APM),

are the direct derivative products of the Arduino based flight controller. APM, launched in 2007 by the DIY Drones, is based on the Arduino open source platform, and has made improvements to multi-threaded hardware, including accelerometers, gyroscopes, and magnetometers integrated into an inertial measurement unit (IMU).

[Meier et al. in 2015] mentioned that PX4 is a hardware and software open source project based on BSD protocol. The purpose is to provide a low-cost, high-performance, high-end, self-driving instrument to academic and industrial development groups. [Meier et al. in 2011] also mentioned that the PIXHawk flight control is the upgrade version of PX4 flight control and has both the PX4 and APM sets of firmware and the corresponding ground station software. PIXHawk currently has a 168MHz operating frequency and the use of integrated hardware floating-point computing core Cortex-M4 microcontroller as the master chip. Also, it incorporates two sets of gyroscopes and accelerometers MEMS sensors which can complement each other. It can also be an external one master two GPS sensors, in the event of failure to automatically switch. [Grabe et al. 2013] provided a novel end-to-end software framework, called Tele-Operation Platform of the MPI for Biological Cybernetics (TeleKyb), for the development of bilateral teleoperation systems between human interfaces and groups of quadrotors (UAVs). [Gültekin et al. in 2016] also uses the pixhawk to communicate with ROS-based sensors via MAVROS messages. After the consideration of functional, compatibility, price, etc., we use PIXHawk flight control and ROS linked by MAVLink with MAVROS in this research.

This research uses a flight control simulator to test the UAV control results. In addition, PX4 provides a more comprehensive framework and examples to assist the developer in building communication between the flight control simulator with ROS.

[Meyer et al. 2012] recommended using the Gazebo and ROS compatible quadcopter simulator complete with simulated wind disturbances. [Meng et al. 2015] proposed a method that uses the ROS and Unity 3D to enable simulated flight of multiple UAVs in a GPS environment. In addition, [Takaya et al. 2016] claimed that after testing code in the simulation, the navigation code could then be used directly without modification. Therefore, we use the Gazebo simulator to test the UAV navigation algorithm before deploying it to a real UAV.

2.3 Visual SLAM

[Davison et al. 2007] provided Mono-SLAM, which applies the traditional EKF-SLAM based on laser range-finder into monocular camera SLAM. This method increases the

computational speed by using guided feature matching instead of the invariant feature matching of EKF-SLAM. However, because the method is based on the filter, it cannot optimize data as the number of trackable feature points in each frame is too small to maintain reliable tracking. When high precision is required, the computational complexity significantly increases too fast. In addition, when the camera is not moving, the output has significant jitter.

[Klein et al. 2007 - 2009] proposed a Parallel Tracking and Mapping (PTAM) SLAM based on keyframes. This method creates a multi-threaded version of SLAM. They separate tracking and mapping into two threads so that the mapping thread can use the bundle adjustment to optimize the accuracy without the consideration of the real-time ability of the tracking. However, the tracking relies on the image feature points if the motion is moving fast, where the tracking is easy to lose.

[Mur-Artal et al. 2015] proposed the ORB-SLAM algorithm based on the PTAM framework. This algorithm has higher stability and accuracy than the PTAM. There are several advantages of ORB-SLAM (over PTAM):

- It has an automatic initial mapping algorithm which can calculate homography matrix and fundamental matrix together through correspondence. Then through a heuristic criteria to determine the current situation, the initial pose can be computed using the corresponding algorithm.
- Except for tracking thread and mapping thread, there is a third loop closure detection thread used to correct scale drift.
- They have designed a more efficient and more suitable keyframe management mechanism for large scale scenes.
- Loop closure detection and relocalization of this method use a more advanced place recognition method (bag-of-words), replacing the small blurry image method in PTAM.
- They use the ORB feature matching instead of the raw patch matching in PTAM.

However, the disadvantage of the ORB-SLAM is that the tracking still relies on image feature points. Since it does not have the uncertainty of model 3D map points, it may cause a cumulative error due to poor initialization.

In the research, visual SLAM is the only localization and mapping algorithm which only depends on the visual odometry. In order to determine the reliability and usability of the

V-SLAM, the GPS, IUM and LiDAR sensors are temporarily closed at the initial stage of the research. Before the research of the SLAM algorithm, the type of SLAM should be determined. Depending on the kind of optical sensor, the V-SLAM can be divided into two main types.

Monocular SLAM is a process where only one monocular camera implements the visual odometry in the SLAM. Most of the open source SLAM methods can support the monocular camera, such as [Javier et al. 2010] EKF monocular SLAM, [Engel et al. in 2014] LSD-SLAM, [Forster et al. 2014] SVO-SLAM and [Mur-Artal et al. 2015] ORB-SLAM. The difference in the above algorithms exists in their way of obtaining features and estimating the depth values. The principle of monocular SLAM is to estimate its motion by moving the camera. Objects in the lens will produce parallax when the camera moves so that the depth of these objects can be estimated. However, the depth information is not absolute, hence the scale uncertainty of monocular SLAM.

The purpose of using a Stereo or RGB-D SLAM is to overcome the scale uncertainty shortcomings of the monocular camera. [Forster et al. 2014] SVO-SLAM, [Engel et al. in 2014] LSD-SLAM, [Mur-Artal et al. 2015] ORB-SLAM, [Endres et al. 2014] RGB-D SLAM and [Labbe et al. 2014] RTAB-Map support RGB-D or stereo SLAM.

As mentioned by [Zanuttigh et al. in 2016], commonly used RGB-D cameras include Kinect V1/V2, RealSense Series (R200, SR300, ZR300, etc), and Xtion Pro Live. However, these RGB-D cameras still have some problems, such as small measuring range, noisy, small field of view, susceptible to solar IR for outdoor use, etc. Therefore, considering the above problems, we use the RGB-D camera (Intel® RealSense™ Camera R200) as the visual sensor at the initial stage of our research.

V-SLAM is a process where a camera moves in a scene to localize the camera pose and map the surrounding environment. It is a complete algorithmic framework as mentioned by [Bailey et al. in 2006]. The flowchart of the framework shows below in **Figure 2.3**. In addition, [Engelhard et al. in 2011] also proposed an RGB-D SLAM method using a similar framework, with an input RGB-D image stream and output 3D point cloud map.

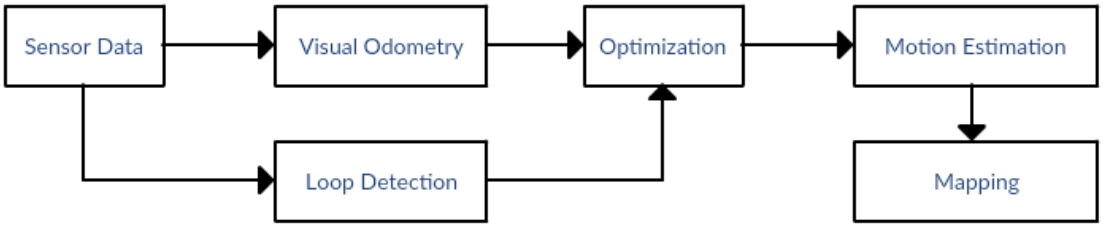


Figure 2.3 The general flowchart of the complete process of V-SLAM.

As shown in the above diagram, the V-SLAM includes following steps as mentioned by [Bailey et al. in 2006].

- Sensor Data Acquisition.

In V-SLAM, this step is the collection and pre-processing of sensor data. It also synchronizes the acquired information.

- Visual Odometry.

To estimate the camera's relative motion between two successive images and the local map. This step can be seen as a front end of the SLAM.

- Optimization.

This step will accept the information of camera motions from visual odometry and loop closure detection measurement. They will be optimized to get a global camera trajectory and map. [Labbe et al. 2014], [Mur-Artal et al. 2015], [Endres et al. 2014], etc. V-SLAM method mentioned that the optimisation step is necessary, where this step can be also seen as a back end of the SLAM.

- Mapping.

According to the optimized trajectory estimation result from the back end, this step will follow the requirements of the research to establish a suitable map, mainly divided as a metric map and topological map.

- Loop Detection.

Usually uses a dictionary to detect whether the camera has reached a certain location. [Gálvez-López et al. 2012], [Labbe et al. 2013], and [Kähler et al. 2016] mentioned the dictionary Bag-of-Words loop detection method.

This framework and their algorithms enable the V-SLAM's localization and mapping to work in a static constrained environment. The following sub-sections of the paper show some relevant SLAM studies from the above steps.

2.3.1 Visual Odometry

Visual odometry is one of the methods of visual navigation which can be traced back to 1980, where [Moravec in 1980] used a mobile camera to obtain visual information of a robot and complete indoor navigation. In 1987, [Matties et al. 1987] presented the concept of visual odometry. They designed a theoretical framework which included feature extraction, feature matching, tracking, and motion estimation. Most of the recent visual odometry systems still follow this traditional framework. Initially visual odometry was based on constrained indoor environments and so development of vision-based navigation research had stalled for a while by [Alismail et al. 2009] because of the high computational complexity, low accuracy, expensive sensors, etc. A decade ago, with the development of faster computer hardware and better algorithms, more image processing tasks were running in real-time, where for example in 2004, [Nister et al.] designed a real-time visual odometry system that implemented outdoor mobile robot navigation.

1) Monocular Visual Odometry

[Scaramuzza et al. 2011] mentioned that monocular visual odometry used the ego-motion information of two successive frames to measure the camera pose and position by extracting feature points and for subsequent picture alignment, minimizing the photometric error. [Forster et al. 2014] presented a semi-direct monocular visual odometry that directly operated on pixel intensities, where repetitive, and high-frequency texture increased robustness. [Lee et al. 2012] also designed a real-time 3D ego-motion tracking system using an IMU and monocular camera. Monocular camera estimates of the ego-motion rely on the motion triangulation measurement. If the camera does not move, the pixel position cannot be estimated.

2) Stereo Visual Odometry

[Scaramuzza et al. 2011] mentioned that stereo visual odometry has multiple visual sensors. Different from monocular, the stereo camera can estimate depth in both motion and at rest. The 3D coordination of feature points can be gained directly. [Persson et al. 2015] involved some monocular techniques to develop a stereo visual odometry system on the KITTI odometry benchmark. However, the translation estimates of high-speed

scenarios with far image areas are poor. In addition, the stereo camera has a complex calibration setup, and the depth range is limited by baseline and resolution of cameras.

3) RGB-D Visual Odometry

RGB-D visual odometry uses a color sensor and a depth sensor, such as infrared light [Dryanovski et al. 2013], sonar [De Pasqua 2014], LiDAR [Zhang et al. 2015], etc. In general, the Intel R200 has relatively accurate and detailed depth information within approximately four meters according to the [Intel RealSense Datasheet]. LiDAR has the potential for accurate long range depth data, but a good LiDAR can be expensive. [Fang et al. 2015] compared several real-time RGB-D visual odometry methods and divided them into three categories.

1) Image-based, which has RGB and depth data type. There are three methods for this category: Libviso2, a fast algorithm for computing the 6 DOF motion of a moving mono or stereo camera. Fovis, a visual odometry method that estimates the 3D motion of a camera using a source of depth information for each pixel. And dense visual odometry methods, which can fully exploit both the intensity and the depth information provided by RGB-D sensors. [Fang et al. 2015]
2) Depth-based, which has point cloud and depth image data type. There are two approaches for this category. One is iterative closed point (ICP) method for point cloud and the other is range flow method for depth image. However, the point cloud-based method may be too slow to ensure real-time visual odometry. [Fang et al. 2015]
3) Hybrid-based, which has also colour and depth data type. Three methods for the hybrid-based type, one is real-time local visual feature boosted normal distribution transform (NDT) method for RGB-D odometry estimation. A visual features and depth information based visual odometry which is used to align 3D features against a global 3D feature map. And depth enhanced monocular odometry which considered both features with and without depth information. These methods are combining image-based error metrics and depth-based error metrics into one integrated error metric to find the optimal transform. [Fang et al. 2015]

[Scaramuzza et al. 2011] also proposed a method to achieve visual odometry, where it estimated the ego-motion of a robot and only considered the local consistency of the

trajectory. Visual odometry is used to study the relationship of image frames. First of all, the feature points of each frame are extracted. Then these feature points are used to estimate the movement of the camera and the spatial position of the feature points. After removing significant noise by RANSAC, the pose and localization information can be obtained. Moreover, they compared properties and performance of many potential feature detectors, such as Harris, FAST, SIFT, SURF, etc. In fact, [Rublee et al. 2011] considered three main type of feature detector, SIFT, SURF, and ORB. Based on their comparison, we chose ORB descriptor as our feature point detector, because it is two orders of magnitude faster than traditional SIFT and SURF. Some experiment results for 2D image feature point detection is shown in **Table 2.1** below from [Rublee et al. 2011].

Table 2.1 The average times of same features point number of 24 640x480 images.

Detector	**ORB**	**SURF**	**SIFT**
Time per frame (ms)	15.3	217.3	5228.7

Furthermore, [Geiger et al. 2012] presented a famous dataset and benchmark called KITTI Vision Benchmark Suite. This dataset shows a superior performance for outdoor autonomous driving. Their data was recorded from one laser scanner, two of each of color cameras and grayscale cameras, and ground truth from an inertial navigation system (INS). The proposed dataset is useful for testing monocular visual odometry, stereo visual odometry, and LiDAR visual odometry algorithms. The current best visual odometry/SLAM method is a LiDAR-based method (V-LOAM) from [49] with only 0.68% translation error and 0.0014 degree/meter rotation error in the KITTI odometry benchmark rank. They solved fast motion and warranted low-drift problems through the combination of visual odometry and LiDAR odometry. These solutions are able to allow the system to have a more accurate and more robust motion estimation method.

2.3.2 Optimization and Loop Closure Detection

However, visual odometry has an accumulative error which causes drift problems. For example, if there is 1° error between two frames, other frames after these two frames also have the 1° error as well. This optimization step is used to deal with the visual odometer results of the noise problem. In general, there are two types of method to solve this problem. One is the early method via a filter [Durrant-Whyte et al. 2006], such as Extended Kalman Filter [Bailey et al. 2006], Unscented Kalman Filter [Martinez-Cantin

et al. 2005], Particle Filter [Choi et al. 2013], etc. This method is an iterative approach, so the computational requirement increases with increasing collected data.

In past decade, researchers began to work on a solution using structure from motion [Koenderink et al. 1991]. This solution introduces bundle adjustment [Triggs et al. 1999] into the SLAM optimization. [Hartley et al. 2005] proposed a method that uses bundle adjustment (BA) algorithm to reconstruct a 3D map. It is initialized via RANSAC and estimates a set of 3D points and camera poses by minimizing the reprojection error. However, BA is a non-linear algorithm that often leads to a local optimum instead of the global optima. Thus, BA is often used to refine a rough trajectory to a more accurate model as the last step of SLAM.

Graphic optimization is not an iterative process. It needs only to consider all the information in the past observations. Through the optimization, the accumulative errors are evenly divided into each observation. In the V-SLAM, the bundle adjustment is usually represented as a graph-based non-linear optimization [Grisetti et al. 2011], such as General Graph Optimization (G2O), etc. The graph-based optimization can represent the optimization problem intuitively. Also, it can be solved quickly via sparse algebra. In addition, it has a more convenient expression for loop closure detection in the SLAM optimization. In fact, ORB-SLAM, RGB-D SLAM, and RTAB-Map implement the non-linear graph optimization via bundle adjustment and loop closure detection.

An alternative to loop closure is using fiducial markers in the scene. When the camera detects a certain marker, the robot position can be identified through the camera projected image of the fiducial marker. However, in the research in this thesis, the working environment is assumed to be a forest. In the ORB-SLAM2 [Mur-Artal et al. 2015], the BA algorithm is applied with loop closure to enable more accurate reconstruction. When key frames are culled, BA discards outlier observations to ensure that the map contains few outliers. In addition, an existed bag of words model, DBoW2 [Gálvez-López et al. 2012], is used to store and retrieve features of each frame.

2.3.3 Mapping

RGB-D SLAM mapping is the process of generating a map reconstruction from camera collected RGB-D images and optimized camera motion trajectory. In general, the map can be divided into two main classes, metric map [Nieto et al. 2004] and topological map [Thrun et al. 1998]. [Nieto et al. 2004] mentioned that the metric map mainly emphasizes the exact location of the objects on the map. There are mainly two types, sparse and dense.

The sparse map only needs landmarks to be shown, whereas the dense map focuses on reconstructing all objects. Therefore, the sparse map is often used to localize camera position, and the dense map is used to navigate.

Compared with the metric map, the topological map emphasizes the relationship between each small area of the map. It is more like a mathematical graph, formed by nodes and edges. As shown in **Figure 2.4**, the topological map is a sparser representation method than a sparse metric map. There are some map description examples for an RGB-D camera which include 2D grid-based map, 2D topological map, 3D octo-map, 3D point cloud map (sparse, semi-dense [Engel et al. 2013] or dense) as shown in **Figure 2.4**.

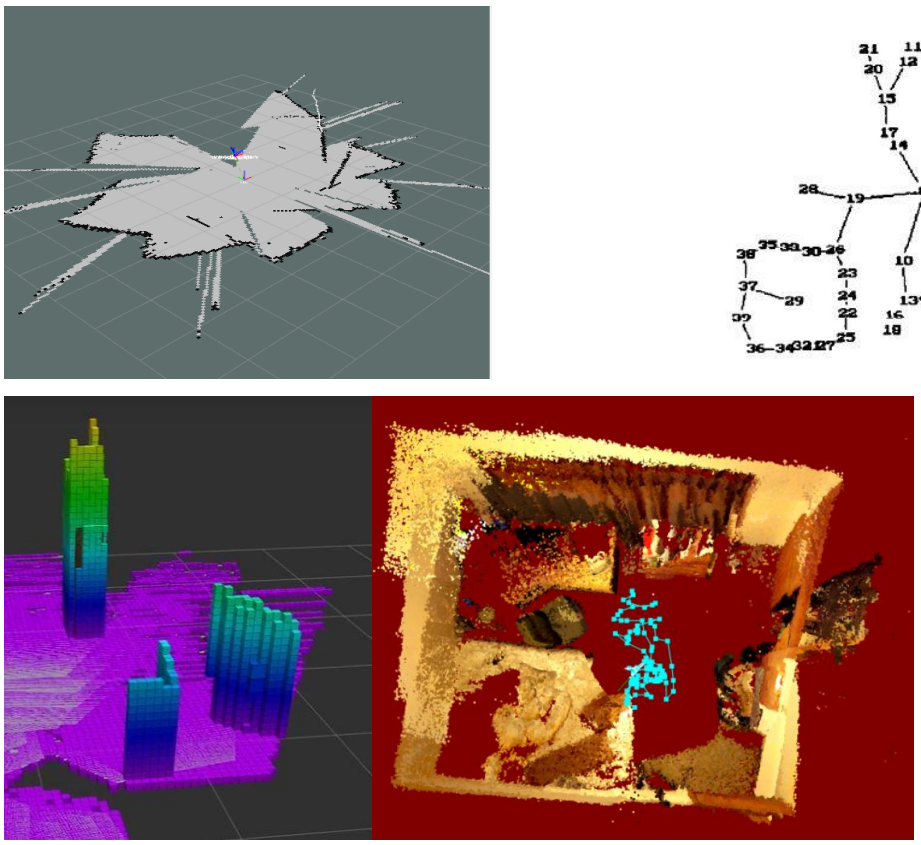


Figure 2.4 The top-left image shows the 2D grid-base map. The top-right shows the 2D topological map, image from [Thrun et al. 1998]. The bottom left image shows the 3D octo-map, and the bottom right image shows the 3D dense point cloud map.

2.4 Object Detection and Localisation

Because a 3D dense point cloud map is just a set of disordered points, it cannot be used to help robot navigation. In the research, we need to detect desired objects and locate them in the point cloud map. The two primary objects which need to be detected are ground and trees.

2.4.1 Ground Removal

Because the global point cloud map includes the ground of the surrounding environment, the ground points can be removed to reduce the consumption of computing resources. [Maguya et al. 2013] mentioned that most of the ground detection algorithms based on an aerial LiDAR scanned point cloud depend on interpolation, morphological, slope, and a hybrid method that combine previous three methods together. Actually, [Moskal et al. 2011] proposed a ground removal method based on the slope for terrestrial LiDAR scanning point cloud. They put the point cloud map into square voxels. For each square voxel, when the slope of the adjacent point voxel is higher than a certain threshold, it will be considered as a non-ground point. However, the slope method has a bad performance on steep terrain. In fact, the threshold setting of the method has high limit application environment. In order to overcome the problem, [Sithole et al. 2001] proposed an improvement slope ground detector, where the threshold changes in the slope of the terrain.

Most of the slope ground detection algorithms can only process two points in each iteration. The interpolation ground filter can handle all points at once. [Mongus et al. 2012] proposed a ground removal method based on the interpolation method. The method uses a plane spline interpolation surface to approximate the ground iteratively. [Maguya et al. 2013] also proposed a ground filter with the interpolation method based on the ground plane surface assumption. However, the result of their paper also mentioned that the method performs poorly with low-density point clouds.

[Yu et al. 2015] proposed an efficient ground filter method based on upward region growing. The method applied the square voxels to increase the computing speed. Then, the method searched nine upward voxels to determine whether there was an object on the ground. They clustered the different height points with the lowest point of the point cloud map to separate the ground with the objects on the ground. This method has good performance on the uneven ground. [Wang et al. 2016] also applied the method in a tree detection research project. However, according to their results the method cannot be used for steep terrain.

To sum up, the method proposed by [Yu et al. 2015] has the best performance for the ground removal method based on point cloud maps. Therefore, the region growing segmentation algorithm for the point cloud is the first considered candidate in our research.

2.4.2 Tree Detection

After the ground detection, a tree segmentation method can be used to extract and localize the trees above the ground. [Dalponte et al. 2011] proposed a tree detection method that rasterizes the point cloud data through a region growing segmentation method. [Yao et al. 2012] also proposed a method watershed segmentation method to rasterize the point cloud map collected from aerial LiDAR scanning. Because the point cloud map is collected from high altitude based aircraft or UAVs, these two approaches often ignore the small trees that grow under the big trees. However, our purpose is to reconstruct 3D point cloud map with the RGB-D camera under the tree canopy area and so the above two methods are not suitable for the research.

Other tree detection methods include Toolbox for LiDAR data Filtering and Forest Studies (TIFFS) by [Chen in 2007], Forest Service Pacific Northwest Research Station's FUSION by [McGaughey et al. 2012] and layer stacking method by [Ayrey et al. 2017]. These previous works all focus on LiDAR data. However, LiDAR data only contains depth information. Although LiDAR has good accuracy, the current high price of high-quality LiDAR devices is prohibitive for extensive use in small industrial drones. Also, these three methods collect point cloud data via an aerial surveying approach whereas we are collecting data below the canopy, close to the ground.

[Yu et al. 2015] claimed that the trees could be extracted via a normalized cut method to separate them from the ground. Their method presents the point cloud map via square voxels. It performs well for separate different trees. However, the voxel finding iteration uses too much computing resource for real-time processing because it needs to restore all weight matrices for the clustering. In addition, the cluster number of the algorithm should be pre-defined for the segmentation. Finally, they recommended a Euclidean clustering segmentation method to extract trees. This method is to segment point clouds based on the pre-defined Euclidean distance threshold to its adjacent points. In the case where the point cloud has a relative average density, we can set a suitable threshold to ensure tree segmentation results perform well.

[Shao et al. 2014] mentioned a tree trunk detection system based on LiDAR to allow a semi-autonomous robot can felling a tree. They used LiDAR to generate a point cloud map. Then, they detect trees through searching normal vectors horizontally. Finally, they cluster the global point cloud map and segment tree trunks through fitting cylinders via random sample consensus estimator and radius range of tree.

As shown in **Table 2.2**, these previous LiDAR methods, the layer stacking method has a better detection rate than the other two methods. Therefore, we design a novel method to detect trees from our point cloud (from RGB-D webcam, not LiDAR) based on layer stacking and Euclidean clustering segmentation.

Table 2.2 The tree detection rate from three different methods, TIFFS, FUSION, Layer Stacking and horizontal directed normal vectors detection.

	TIFFS [Chen 2007]	**FUSION [McGaughey]**	**Layer Stacking [Ayrey et al. 2017]**	**Normal Vectors [Shao et al. 2014]**
Detection rate	42%-76%	32%-59%	66%-89%	50%-100%
Commission error	12%-65%	0%-49%	24%-53%	N/A

2.5 Navigation

Path planning can be based on the following three approaches:

1) Template Matching Method

   The path planning method based on template matching uses the existing and prior information to generate a model library. Each template contains environment and route information. When the current route and environment information is compared with the previous template in the library, an optimum matching template can be generated. This method has good performance in constrained environments [Liu et al. 2004]. Furthermore, some researchers proposed methods that combine template matching with machine learning. [Ram et al. 1997] combined online cases matching with reinforcement learning to increase the adaptive performance of the robot. Moreover, [Arleo et al. 2004] combined environment templates and neural networks to optimize paths. However, experience is necessary and this method mainly focuses on a constrained static environment so that when the environment changes, the template was difficult to match in the library.

2) Artificial Potential Field Method

   Artificial potential field path planning assumes robots move in a virtual potential force field. Obstacles make repulsive forces to the robot, and the target makes an attractive force to the robot. The repulsion and attraction control the robot to avoid

obstacles and reach the target position. [Ge et al. 2002] introduced the relative position and velocity of the robot with targets and obstacles as attractive and repulsive potential functions respectively. The method was applied to a footfall mobile robot path planning with a satisfactory result. [Jaradat et al. 2009] and [Faisal et al. 2013] both combined the fuzzy logic and the artificial potential field and presented a fuzzy artificial potential field algorithm. In addition, they also combined them with robot motion models and presented a completed mobile robot path planning and control method.

However, the artificial potential field method has a significant possibility of generating a local minimum point. Also, the design of attractive and repulsive field has uncertainties. Moreover, when there are many obstacles, computational complexity is also a problem.

3) Mapping Method

Map building path planning spills the surrounding area of a robot into different grid space through the obstacle feedback of sensors. Via the obstacle distribution situation in the grid spaces, an optimal path can be planned. [Oh et al. 2004] presented a robot autonomous navigation approach based on the general Voronoi diagram (GVD) in a virtual environment. GVD contains a set of the line segments and parabolic segments. The method can reduce the probability of collision with an obstacle. [Pan et al. 2012] used the grid method to plan a path. The method requires the surrounding space to decompose into a set of connected and non-overlapping cells. From these cells, there is a search for a non-collision path. To improve the performance of map building route planning, many people use it with other path planning methods, such as [Araujo in 2006] who presented the ART neural network map building method.

2.6 Summary

The chapter illustrates the background and related studies of the UAV flight controller with a simulator based on ROS, the visual SLAM framework, and object detection. ROS is the most suitable software framework to support robotics research and for the flight controller (PX4) and simulator (Gazebo) it has better compatibility and repeatability. According to the previous research, the ORB feature extraction method has better performance than SURF and SIFT. The optimization based on the graph is a more efficient way to optimize the camera trajectory through loop closure detection using a

binary dictionary. A dense point cloud should be used to detect trees rather than a sparse map. Finally, the point cloud slicing method can improve tree cylinder fitting in a 2D slice plane better than fitting directly in 3D space.

3 METHODOLOGY

In this **Chapter I** propose a novel approach that controls a small UAV to automatically navigate to one of the nearby target trees in the forest. To enable this goal, I propose two methods to detect trees. One is to use geometric model fitting to detect trees and get the depth value of the trees from the depth image of each frame. According to the estimated camera pose, the specific world ordination of the detected trees can be calculated. Another approach is to establish the surrounding three-dimensional point cloud map, and then through a 3D point cloud slice/cluster/segmentation method, detect the specific location of the trees. Finally, both these tree detection methods are merged to estimate a more accurate position of the trees.

In addition to tree detection and location, this chapter will also describe how to control the UAV to fly to the nearest unvisited tree after acquiring the pose of the drone and the position of the tree. As part of a large project based on ROS, the ROS topic and service are used for asynchronous communication and synchronous communication between one of this project's ROS nodes with others. In this research, the tree positions are communicated by a ROS service and the UAV navigation system ROS node is used to communicate with a Pixhawk flight controller (PX4) by MAVlink. The navigation-communication system is tested in a Gazebo simulator.

In general, there are four steps to control a small UAV to navigate to a target tree - detection, location, tracking, and navigation. A flow chart of the implemented algorithm is presented as **Figure 3.1**. The following sections of this chapter will describe each box shown in **Figure 3.1**.

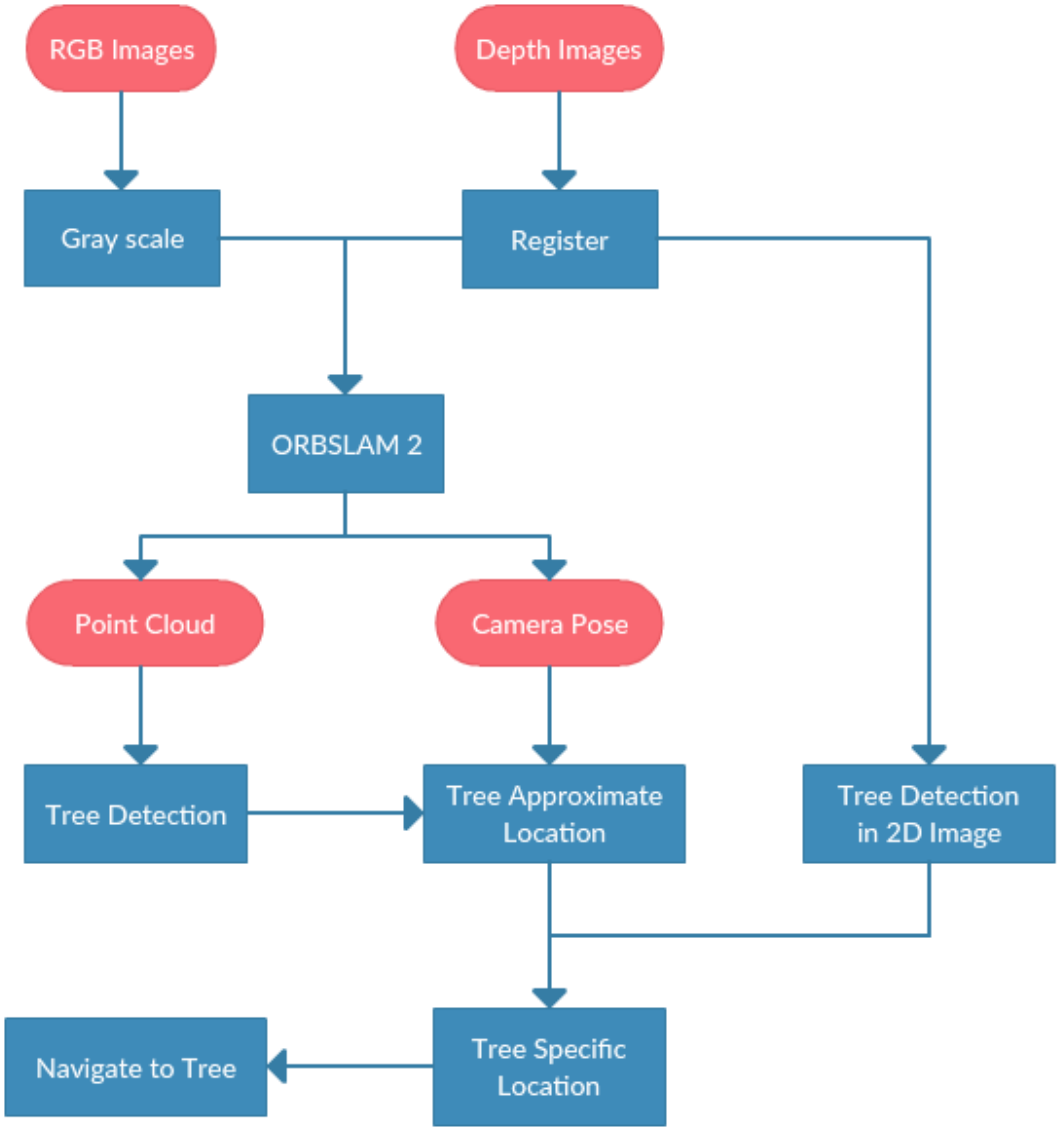


Figure 3.1 The flow chart of the tree detection and navigation algorithm. RGB (gray-scale) and depth (registered) sequences are required to estimate camera pose and generate point cloud by our modified ORBSLAM2. Trees are detected after the surrounding point cloud is generated with loop closure and full bundle adjustment. Related approximate positions between trees and camera can be estimated. The specific tree's location can also be adjusted to the approximate position of trees using the current RGB-D frame in real-time when a detected tree (from point cloud) is within the field of view of the camera. Finally, the system sends a message as a ROS topic to control a UAV toward a target tree.

3.1 Data Acquisition

A high-quality point cloud map is necessary for this research. Therefore, we have a higher demand for the data collection quality, especially the collection of depth value data. This

section will describe how to determine the most appropriate data acquisition method for the research project from both software and hardware aspects.

3.1.1 Hardware Environment

Currently, there are two different outdoor autonomous environmental navigation solutions. One is the use of LiDAR solutions. Google [Waymo] and Uber [Brewster in 2015] use this solution to solve their automatic driving problems. [Reutebuch et al. 2005] mentioned that the principle of LiDAR is to calculate the time of travel (ToT) of this laser, the distance between the transmitter and reflector is determined. Moreover, according to the principle of the known speed of light, the propagation time will be converted to the distance between the objects with LiDAR. Therefore, LiDAR has large advantages in the accuracy of the range. However, this solution is very expensive. For example, the price of Velodyne HDL-64E LiDAR (64 lasers), on the top of the Google unmanned car prototype, is 75000 US dollars mentioned by [Amadeo in 2017]. Even a 16 laser LiDAR (Velodyne VLP-16) sufficiently light for drones costs 8000 US dollars – and even the size and weight of this LiDAR is not suitable for use on a small drone.

Figure 3.2 The top images show a Velodyne HDL-64E LiDAR sensor and a Bumblebee2 stereo camera. The bottom image shows a Kinect V1 and a Kinect V2, and an Intel RealSense R200.

A much cheaper and lighter solution than the previous one uses multiple closer range sensors (depth camera) instead of a LiDAR. In general, there are three main types of depth cameras. As shown in **Table 3.1**, the comparison of these three kinds of depth cameras indicates which type is most suitable for specific tasks. This section will briefly introduce these three depth cameras.

Table 3.1 Comparison of these three common depth detection technologies, structured light, time of flight and stereo camera.

Type	**Structured Light**	**TOF**	**Stereo Camera**
Resolution	Medium	Low	High
Accuracy	Medium	Medium	High
Frequency	Medium	Fast	Medium
sunlight ok	Low	Low	High
Size	Large	Large	Small
Cost	Medium	Medium	Low
Development difficulty	Medium	Low	High

- Structured light, is based on optical coding and triangulation. An infrared projector projects the known infrared pattern into the scene. Then the depth information of the sensor is determined by the deformation of the pattern captured by another infrared CMOS imager. Consequently, this sensor cannot be used in outdoor environments in sunlight. A representative product is Kinect 1.
- Time of flight (TOF), which based on the known speed of light, measures the depth value through calculating the time delay using phase shift of a modulated near-infrared light signal emitted to a surface and return to the receiver. A representative product is Kinect 2.
- Stereo camera, which simulates human visual principles, uses two or more passive visible (or infrared) sensors to estimate the distance. It observes an object from two or more points to obtain the images under a different angle of views. According to the matching relationship between the pixels, the depth value of each matching pixel is calculated by the offset between pixels from different monocular camera images based on the triangulation principle. Examples are Intel RealSense R200 and Bumblebee2 1394a (passive visible light).

Since these vision sensors need to be mounted on a small UAV, the following points must be considered:

- Ok in sunlight (High Request) – The project scenarios are applied to the modeling of trees in the outdoor environment. Therefore, the request of the sensor for light interference is high.
- Size & wieght (Small Request) – The sensor needs be mounted on a small UAV. The size of the sensors need to be as small and light as possible to increase the endurance of the UAV.
- Cost (Low Request) – Consider the hardware costs (less than 1000 US dollars) for visual sensors on the research. Although, in this research there is only one camera mounted on the front of the UAV, at least Three cameras are used to monitor the current UAV states in the future application (up/down/forward facing).

After consideration of the above three factors, in this research, Intel® RealSense™ Camera R200, a long-range stereo vision 3D imaging module is used as an RGB and depth sensor. This camera has small dimensions (101.56mm length, 9.55mm height, 3.8mm width as shown in **Figure 3.3**) that can be flexibly applied to a wider range of environment [R200 official Datasheet]. In **Figure 3.3**, there are two infrared cameras and a color camera.

The camera can provide color, depth, and infrared video streams. The maximum resolution of the color camera is 1980 x 1080 with fixed focus. The field of view is vertical (43°+/-2°) and horizontal (70° +/-2°). Moreover, the maximum resolution of the registered depth camera is 640 x 480 with fixed focus as well. The field of view is vertical (46°+/-5°) and horizontal (59° +/-5°). Although the depth stream supports 30 and 60 frame rates, 30 depth stream frame rates are used to save computing resource in this research. In addition, the color image resolution is resized from 1980 x 1080 to 640 x 480 to make the color and depth streams the same resolution and frame rate, which allows the two different streams to be better synchronized.

In this research, the proposed methods are run on a Intel Core i7 CPU with 4×2.4 GHz core frequency based on AVX2 64bit instruction set. The device has 16G (8GB×2) memory capacity with DDR3 16MHz. In addition, it has NVIDIA GeForce GTX 765M video card with 2GB video memory and 768 stream processors.

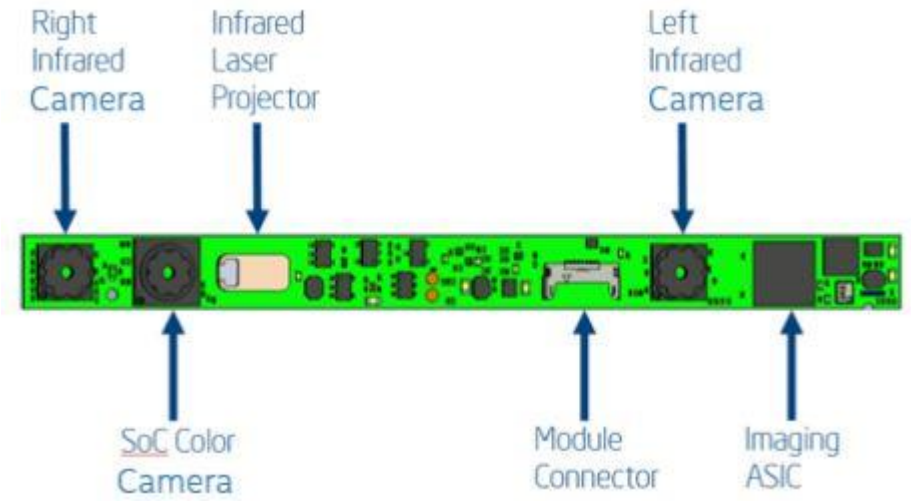


Figure 3.3 The front view of Intel® RealSense™ Camera R200 without reinforcement frame to expose components. The image comes from [R200 official Datasheet].

3.1.2 Software Environment

The system is built based on Linux Ubuntu 16.04.2 LTS (Xenial Xerus) which has 4.4.0 Linux kernel version (which passed all tests and implemented algorithm under the environment). In order to make this study suitable for a UAV with onboard SDK, an open source ROS (Kinect), flexible framework for writing robotic software, is the core sub-system used to control the communication between sensors, system (Ubuntu/PX4), and UAV propeller motors.

Several libraries, or ROS packages are used in the research - librealsense (Linux driver for RealSense R200 camera), realsense_ros (ROS package for the R200), OpenCV, CGAL, g2o, Eigen, etc. As mentioned before, the R200 Camera Linux Driver – librealsense and the most of popular exist 2D and 3D image or point cloud processing libraries are implemented by C++. Therefore, we decided to use C++ as the main programming language in this research. And we also used python as scripting language if necessary.

The camera pose estimation tracking is visualized by a development library called Pangolin, which can manage OpenGL display and abstract input footages. Furthermore, our team decided to use PX4 as the flight controller for our UAV. Firstly, PX4 flight controller can be developed in different operation systems include Mac OS, Linux, Windows. And PX4 has a very detailed user guideline. And it has good compatibility with ROS through MAVROS interface. In addition, the PX4 can also support the current version Gazebo simulator to test our UAV before using real machine.

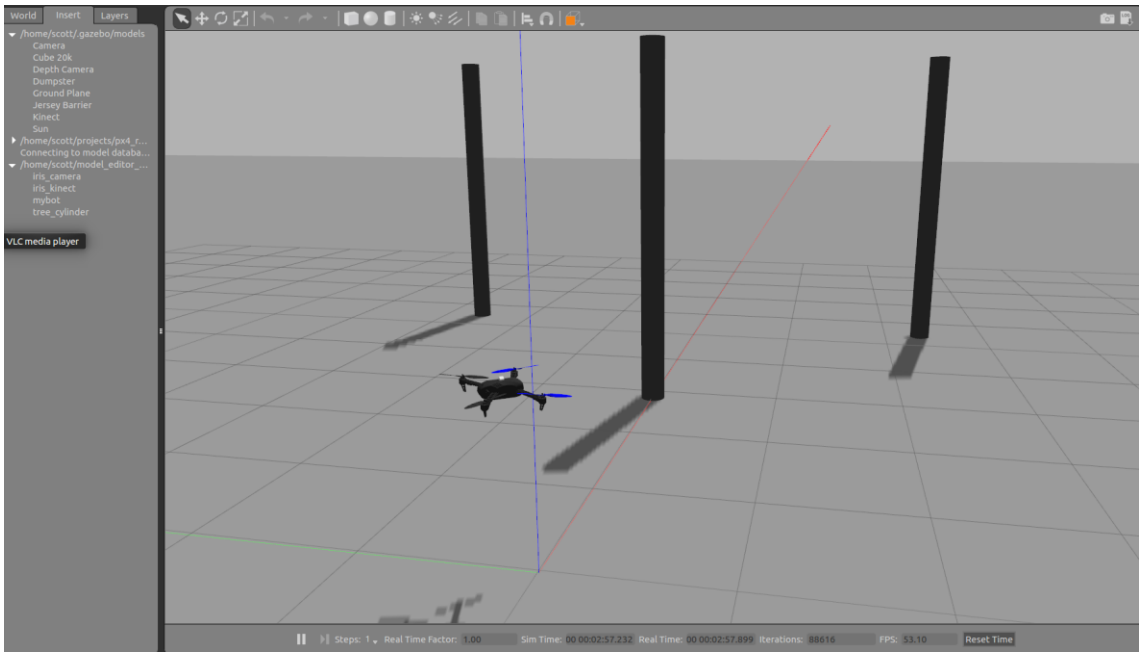


Figure 3.4 Gazebo 7 four-axis UAV simulation.

3.1.3 Depth Map Denoise and Hole Filling

In the process of acquiring an image stream, due to the influence of the equipment and the external environment, noise is unavoidable. The Intel® RealSense™ Camera R200 measures depth value via two infrared cameras. By observing the acquired depth image **Figure 3.5**, there is not only noise, but there are also some holes that mean the device did not get the depth value in the hole area. Therefore, before using these depth images to do a SLAM algorithm, the collected image stream needs to be pre-processed through denoising and hole filling.

(a) (b) (c)

Figure 3.5 R200 camera depth images. (a) The image is raw depth image in the indoor environment. (b) This image is the registered depth image. (c) The image is the pre-processed registered depth image.

Intel® RealSense™ Camera R200 can emit infrared light (Class 1, IEC 60825-1:2007 Edition 2) which is nominal wave (859nm wavelength) on a laser projector. The camera can be used in low light conditions when the parameters pre-setting of the camera laser transmitter and receiver is for a low light condition. But if the camera want to be used in sunlight environment, the parameters of the transmitter and receiver need be reset. Then, the two infrared cameras receive the reflected infrared laser from the object surface to generate raw infrared data. Finally, the two infrared cameras can be seen as a stereo camera system. The stereo reflected infrared system generates the raw depth image stream. However, there are circumstances under which the infrared laser cannot be reflected from the surface of an object, and so the depth value will not be detected. There are two main reasons for this:

- The special surface material, for example, glass, mirror, absorbent infrared material, etc.
- The special structure which cannot reflect infrared laser.

The second reason for the noise is the limitation of the device. The R200 camera can generate noise and holes even for infrared reflection from a simple plane (wall), as shown in **Figure 3.5**. However, there are enough depth values to determine the location of a tree after denoising and hole filling. The following two sections will describe a method to reduce noise and holes in such raw depth images.

In general, there are two main denoising filter approaches in the field of image processing, the linear filter such as box filter, mean filtering Gaussian filtering, and the non-linear filter such as median filter and bilateral filter. Linear filters are often used to eliminate unwanted frequencies in the input signal or to select the desired frequency from many frequencies. In general, there are several kinds of linear filter.

- A low-pass filter that allows low frequencies to pass.
- A high-pass filter that allows high frequencies to pass.
- A bandpass filter that allows a range of frequencies to pass through.
- A band-stop filter that prevents a certain range of frequencies from passing through and allows other frequencies to pass.
- An all-pass filter that allows all frequencies to pass through, just changing the phase relationship

Box filter and mean filter are two simple linear filters; each pixel of the output image is the average of the pixels of the corresponding pixel of the input image in the kernel window. Mean filter is a normalized Box filter.

Gaussian filtering is a linear smoothing filter that is suitable for eliminating Gaussian noise. Every pixel of the image is weighted by its own and other pixel values in the field. A 2D Gaussian equation can represent as:

$$\boldsymbol{G_0(x,y) = Ae^{\frac{-(x-\mu_x)^2}{2\sigma_x^2}+\frac{-(y-\mu_y)^2}{2\sigma_y^2}}} \quad (1)$$

Where μ_x and μ_y means the peak value in x axis and y axis. σ means the variance per each of the variables x and y. The Gaussian filter is a good filter for the both time domain and frequency domain. In detail, it has five important properties that are widely used in image processing.

- Rotational symmetry of the Gaussian function, which means the filter has the same degree of smoothness in every direction. In the following processing, it will not bias in any direction.
- The Gaussian function is a single value function, which means every pixel value is weighted by its own and other pixel values within its field. Moreover, the weight of the adjacent pixel is changed with the distance between its center pixel.
- The Fourier transform spectrum is singular, which means the Gaussian filter keeps most of the required low-frequency signals when removing useless high-frequency signals.
- Adjustable filter width. It is used to determine the degree of smoothness of image by σ. By adjusting the value of σ, the image features can be selected between over smooth and under smooth.
- Separability, which means the larger scale Gaussian filter can be effectively implemented.

The linear filter is easy to construct and easily analyzed from the frequency response angle. The output value of each pixel depends on the weighted sum of input pixels. However, sometimes non-linear pixels can get better results when an image has outliers occasionally. In this case, when the Gaussian filter is used to blur the image, the noise pixels may not be removed.

Median filtering is a kind of nonlinear signal processing technology which can effectively suppress noise based on ranking statistical theory. The basic idea is to replace the depth (single channel) value of the pixel with the median value of the adjacent pixels. It makes the surrounding pixel value close to the actual value and eliminates isolated noise points. Because it does not depend on the values of the adjacent point that are very different from the typical values, the method can remove the impulse noise, salt, and pepper noise while preserving the edge of the image. The median filter operates similarly to the linear filter when processing an image stream, but the filtering process is no longer a weighted operation. The median filter can overcome the image detail blur of the common linear filter. In addition, this filter is not only very effective for filtering out the pulse interference and image scanning noise. However, it also has a feature that can be used to protect edge information. Therefore, this feature means the filter has good performance in depth image filtering.

In addition, the bilateral filter is a non-linear filter as well. Bilateral filter is an improvement of the Gaussian filter. Compared with Gaussian filter, it has one more parameter, Gaussian variance sigma-d. It is based on the spatial distribution of the Gaussian filter function; the pixel values on the edges are not affected by the distant pixels. It means the filter can also guarantee the reliability of the pixel values near the edge. In the bilateral filter, the value of the output pixel depends on the combination of the weighted values of the adjacent pixel values, as shown below:

$$\mathrm{g(i,j)} = \frac{\sum_{k,l} f(k,l) w(i,j,k,l)}{w(i,j,k,l)} \tag{2}$$

Moreover, the weighting factor $w(i,j,k,l)$ depends on the product of the definition domain kernel $d(i,j,k,l)$ and the value kernel $r(i,j,k,l)$, depth value in this research, they are presented as:

$$\mathrm{w(i,j,k,l)} = \mathrm{d(i,j,k,l)} \times r(i,j,k,l) \tag{3}$$

$$\mathrm{d(i,j,k,l)} = \exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2}\right) \tag{4}$$

$$\mathrm{r(i,j,k,l)} = \exp\left(-\frac{||f(i,j)-f(k,l)||^2}{2\sigma_r^2}\right) \tag{5}$$

Although, the filter can denoise part of noise points in a depth value, depth images from a R200 camera has too many holes, and the issue will affect the quality of 3D point cloud map reconstruction. Therefore, these holes need to be filled to reconstruct a better depth image. Since the depth value in the small adjacent area can be seen as continuous, the

holes can be filled with the effective depth values around the holes. However, if the adjacent point is directly estimated, some errors may occur which estimate that the pixels around the hole are not on the same object. Therefore, in order to avoid the shortcomings of false estimates, an improved weighted bilateral filter is used to estimate the depth values of the hole. This method can not only fill the holes but also keep the edge of the depth image.

3.2 Simultaneous Localisation and Mapping

After the depth and color data collection and pre-processing, a SLAM algorithm should begin to estimate camera pose and reconstruct the surrounding 3D point cloud map. The SLAM solution enables a mobile robot from an unknown location to start moving in an unknown environment. During the processing, it can estimate the position while mapping the surrounding 3D point cloud map based on the position. An accurate 3D map can enable an autonomous robot/UAV to navigate to an exact location in the certain environment. In addition, this research uses a SLAM algorithm only based on visual information from an RGB-D sensor. This method can also be called visual-SLAM (V-SLAM) or RGB-D SLAM.

In this section, the V-SLAM algorithm will be presented in four parts - camera calibration, pose estimation, mapping and loop closure. As the first step to ensure the visual-SLAM has higher accuracy, the research uses a 9×7 black and white checkerboard to measure the intrinsic camera parameters and the camera distortion parameters. The camera pose estimation depends on visual odometry in this research (although the future work of the tree pruning project has more sensor data requirement, such as IMU, gravity, magnetic field, GPS, etc). The expression of the 3D point cloud mapping result is a density point cloud that can satisfy the requirement of this research scene. In addition, in order to avoid the double point cloud errors in the map, loop closure is executed between the 3D point cloud mapping and the next tree detection step. As shown in **Figure 3.6**, the flow chart of the SLAM framework includes pose estimation and mapping. Input data are calibrated color and depth images. The frontend processes extract feature points to estimate the camera motion trajectory. The backend optimization uses a graph optimization method to average the effect from image feature error through bundle adjustment. Finally, the loop closure detection uses bag-of-words (BoW) to match an arrived scene to reduce cumulative error significantly.

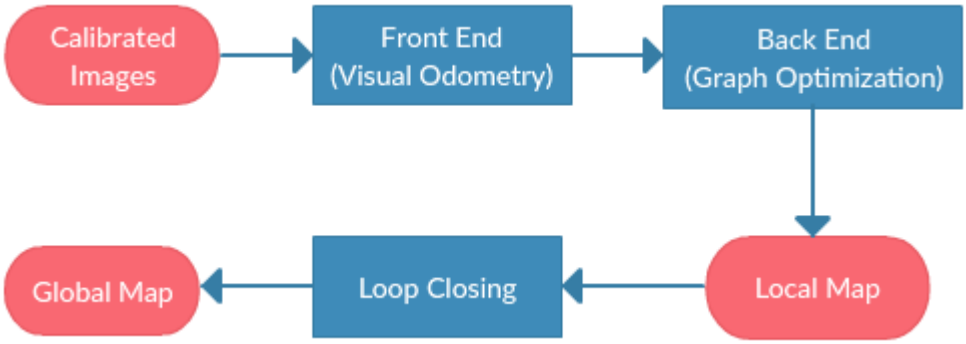


Figure 3.6 The workflow of Visual SLAM.

3.2.1 Camera Calibration

After the depth image and color image are collected from the camera, these two images have to be calibrated to ensure a more accurate registration between them. In this research, because the camera could be seen as a point and the initial position of the camera is at the origin point, extrinsic camera parameters are not generated. The intrinsic camera parameters and camera distortion parameters are required, and are measured by a camera calibration ROS package which mainly uses the OpenCV camera calibration module.

The intrinsic parameters are outputted as a 3 x 3 matrix $\mathrm{K} = \begin{bmatrix} fx & s & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$ which contains 4 parameters in OpenCV camera calibration module, includes focus of the *x*(*fx*), focus of the *y*(*fy*), center point coordinates in x and y axis relative with the imaging plane (*cx* and *cy*). Parameter *s* is the tilt parameter of the coordinate axis, ideally as 0. In addition, the distortion parameters are also generated as 3 radial distortion parameters $(k1, k2, k3)$ and 2 tangential distortion parameters $(p1, p2)$, $D = (k1, k2, p1, p2, k3)$. The camera calibration result is shown in **Appendix A**.

3.2.2 Motion Estimation

As part of V-SLAM, motion estimation and graph optimization play two key roles. This section shows the method to estimate the camera pose through visual odometry and the optimization processing of the visual odometry.

3.2.2.1 Visual Odometry

Visual odometry can get a camera pose information, including position and attitude, by estimating the relative movement of the camera at two different moments. The camera is moving in a Euclidean distance model. A three-dimensional transformation matrix is the

result of the visual odometry camera pose estimation. In order to solve the transformation matrix, two different methods can be used, direct measurement method and feature point method.

The direct measurement method writes all the pixels of the image into a pose estimation equation directly to find the relative motion between frames. For example, in RGBD SLAM, the transformation matrix between two point clouds can be solved using ICP (Iterative Closest Point). They have achieved success. However, the direct method requires more computational complexity, and the image acquisition rate of the camera also has a higher demand as well.

Therefore, to save computing resources and obtain more accurate trajectory estimates, the feature point sampling method is used in the research. In detail, for two frames of footage, the feature points are extracted firstly. Moreover, according to the feature matching of the two frames, the camera's transformation matrix can be calculated. There are some common point feature extract methods, such as Harris Corner, SIFT, SURF, ORB, FAST. The ORB feature point detection method is a compromise choice that considers both computing speed and feature extract quality. For an RGB-D camera, the known depth value of each feature point can directly be used to estimate the motion of the camera. This problem solves the camera pose from a set of feature points and the pairing relationship between them called Perspective-N-Point problem (PnP). Nonlinear optimization can address the PnP issue. The relative position between two frames are figured out through solving the PnP problem by non-linear optimization.

In general, the front-end visual odometry has two parts, feature match and multiple view geometries as mentioned by [Martines et al. 2005], which include Epipolar geometry, PnP, rigid body motion, etc. The feature matching has three steps.

- Feature detection – The first step is to find out the feature points set for each frame.
- Then, for each detected points of each frame, the algorithm extracts their descriptors.
- Every descriptor from one frame will be compared with every descriptor from other frame for scoring them.
- A pair of descriptors with the highest score will be considered as the best match.

In addition, if the feature description can ascertain the direction of the feature, rotation invariance can be achieved. Moreover, if the feature description can determine the scale,

then scale invariance can be accomplished. As one of the best feature extraction and matching algorithms, ORB feature matching is used in the research. The ORB feature detection method uses features from accelerated segment test (FAST) algorithm to detect feature points. The definition is based on the pixel gray value or the pixel depth value of the image around the candidate feature point. The candidate point is considered as a feature point if there is enough pixel value difference between the candidate point with points in its neighborhood. The FAST algorithm can be seen as:

$$\mathrm{N} = \sum_{x \forall (circle(p))} |I(x) - I(p)| > \varepsilon_d \qquad (6)$$

Where $I(x)$ is the gray or depth value of any point on the circumference. $I(p)$ is the gray or depth value of the candidate point. The ε_d is the threshold of the difference of pixel value. If N is greater than the given threshold, then p is a feature point. In this research, the threshold ε_d is 19 and the minimum value of N is 7.

Although the FAST algorithm is a rapid method of extracting features, it has two shortcomings:

- The extracted feature points do not satisfy the multi-scale feature points. Therefore, the ORB feature matching method uses an image scale pyramid to reject some candidate feature points via Harris filter for each of the pyramid levels. In the scale pyramid of the project, we set 1.2 as the scale factor between levels and the number of levels as 8.
- The extracted feature points have no orientation. The ORB feature matching can use the intensity centroid to determine the direction of the feature points.

In detail, if the neighborhood of the feature point is a patch descriptor, then find a straight line between the feature point with the center of the patch. The angle between the straight line and the horizontal axis is the orientation of the feature point.

After we get the feature points, we need to describe the properties of these feature points in some way. The output of these properties is called the descriptor of the feature point. The ORB uses a rotation-aware BRIEF algorithm to compute a descriptor for a feature point. The ORB feature descriptor is formed by the rotation-aware BRIEF algorithm. It can solve the problem that normal BRIEF is very sensitive to the rotation operation. The method is to rotate the matched points in the adjacent field. The coordinates of the matching point after rotation can be used directly when the feature point descriptor is

requested. The combination of oriented FAST and rotation-aware BRIEF makes up the whole ORB feature matching algorithm.

The next step is to solve a PnP problem using the relative position between two frames to estimate an initial camera position via the above ORB feature extract method. If the feature tracking is lost, the system can also try to re-localize the camera motion through RANSAC keyframe iteration. According to the previously generated point cloud and the obtained keyframe images, the camera pose and local map can be generated. Theoretically, the camera motion trajectory can be estimated through the above visual odometry.

3.2.2.2 Optimization Processing

This subsection shows the backend optimization processing of the V-SLAM to ensure the accuracy of the camera localization and mapping. Intuitively, our tree detection approach needs at least one camera to scan 360° around the environment to determine the relative location of the closest tree. However, if an estimated angle is less than the actual angle of even 1°, then all the subsequent track measurements will also be out by 1°. Such consequence of the cumulative error may cause the estimated trajectory to drastically drift. In addition, the output 3D point cloud image quality may not be high, and even appear overlapping point cloud. Therefore, in the V-SLAM, the relative motion between the frames requires a process known as backend processing.

In general, there are two types of visual SLAM backend process method to optimize the estimation of camera motion. An early method optimizes the problem via a filter, such as [Fujii in 2013] Extended Kalman Filter (EKF), [Wan et al. 2000] Unscented Kalman Filter (UKF), [Sim et al. 2005] Particle Filter (PK), etc. When a frame arrives, the relative position between the camera with the detected features can be detected through the observation of the sensor, but there is noise and error. Through the observation equation, the position of the camera can now be predicted. In addition, according to the previously recorded features, a Kalman gain can be calculated to compensate for the effects of noise. Therefore, this is an iteration processing that predicts and renews the position of the camera.

However [Hartley in 2005] mentioned that after the 21st century, SLAM researchers began to learn from the structure from motion problem (SfM) in the backend method and Bundle adjustment was introduced into SLAM algorithm. The bundle adjustment optimization method and the filter method are fundamentally different. It is not an iterative process, but rather the information in all past frames needs to be considered. By

the optimization method, the error will be evenly divided into each frame. Bundle Adjustment in SLAM is often given in the form of graphs, so it is also known as graph optimization. It expresses an optimization problem as a graph to help to understand better and observe. Graph optimization is the mainstream optimization method for the current V-SLAM.

Essentially, graph optimization is an optimization problem. There are three primary factors in the optimization problem: an objective function, optimization variable, and optimization constraint. A simple optimization problem can be described below:

$$\min_{x} F(x) \tag{7}$$

Where x is the optimization variable and $F(x)$ is the optimization function. This is an unconstrained optimization problem description. In fact, most of optimization problems are unconstrained in the V-SLAM. Since the form of $F(x)$ is uncertain, it is necessary to traverse all the zero gradient points to find the smallest as the optimal solution. However, many times the form of $F(x)$ is too complex, the analytic form of the derivative cannot be written, or it is difficult to solve the equation with the derivative zero. This is the reason the early optimization problem always uses an iteration method to continually approach the minimum point. The iterative strategy is mainly reflected in how to choose the direction of descent, and how to choose the step size. There are two methods, Gauss-Newton (GN) method [Bell et al. 1993] and [Ranganathan in 2004] the Levenberg-Marquardt (LM) method.

[Kümmerle et al. 2011] mentioned that the graph optimization expresses an optimization problem as a graph. The graph is composed of vertices and edges. In graph theory, hypergraph means a graph in the connection of two or more vertices. The SLAM problem can be expressed as a hypergraph. The core of SLAM is to calculate the motion trajectory and surrounding map of the camera based on the existing observations. Assuming the time is t, the camera at the position p_t with the sensor for an observation to output a data d_t. The observation equation is:

$$d_t = \mathrm{h}(p_t) \tag{8}$$

Due to the existence of the error, d_t can not exactly equal to $\mathrm{h}(p_t)$. There is an error equation:

$$e_t = d_t - \mathrm{h}(p_t) \tag{9}$$

Therefore, the graph optimization problem in SLAM can be expressed as:

$$\min_{p} F_t(p_t) = abs(e_t) \tag{10}$$

In general, the process of the graph optimization has six steps.

1) To select the type of node and edge in the graph to determine their parameterized form.
2) To join the original node and edge into the graph.
3) To select the initial value, start iteration.
4) In each iteration, the Jacobian matrix and the Cypriot matrix of the current estimate are calculated.
5) To get the gradient direction by solving the sparse linear equation
6) To iterate with GN or LM. If the iteration ends, the optimization value is returned.

In the research, the bundle adjustment of the optimization backend is implemented by a graph optimization framework, General Graph Optimization (g2o). The framework has already implemented steps 3 to 6 of the above steps. In ORB-SLAM, a graph method, essential graph, is used to optimize the global loop closing.

3.2.3 3D Point Cloud Reconstruction

Point cloud map is one of the representations of the 3D map. It means that a set of points represents the coordinates and distribution of the objects in the space. By plotting a mass of points in the space and using these points to form a data set, a 3D model is created to represent the surface properties of the scene. The 3D RGB-D camera, such as RealSense R200, can collect depth image that allows us to directly get the relative camera position of the target object. In order to detect our objects, trees, in 3D map, a global point cloud map is necessary. In addition, because the RGB-D camera has a narrower view of the field than LiDAR, and the canopy of the pine tree may affect the flight of UAVs, we only allow the UAV to fly below the canopy. Therefore, the reconstructed point cloud map contains only the trunk and the ground as shown in **Figure 3.7**.

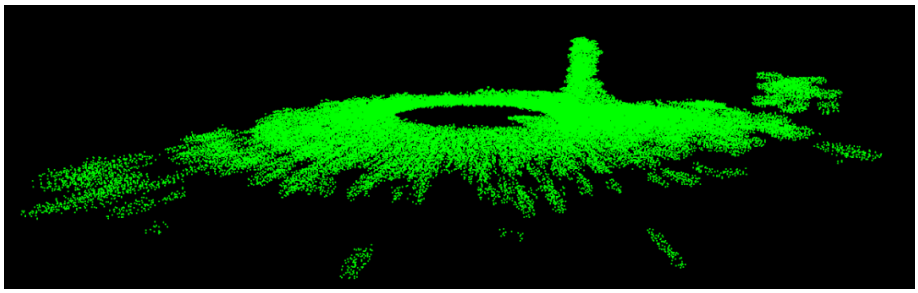


Figure 3.7 The point cloud image only contains ground and tree without the canopy.

In the processing of the 3D point cloud mapping, the expression of point cloud map has two different methods, sparse and dense. Sparse point cloud has less information and data for the target object surface detection. It cannot be used to detect and locate an object accurately. From the previously related studies, [Mur-Artal et al. 2015] proposed a method, ORB-SLAM, which can generate a sparse point cloud map with a rough map outline as shown in **Figure 3.8**.

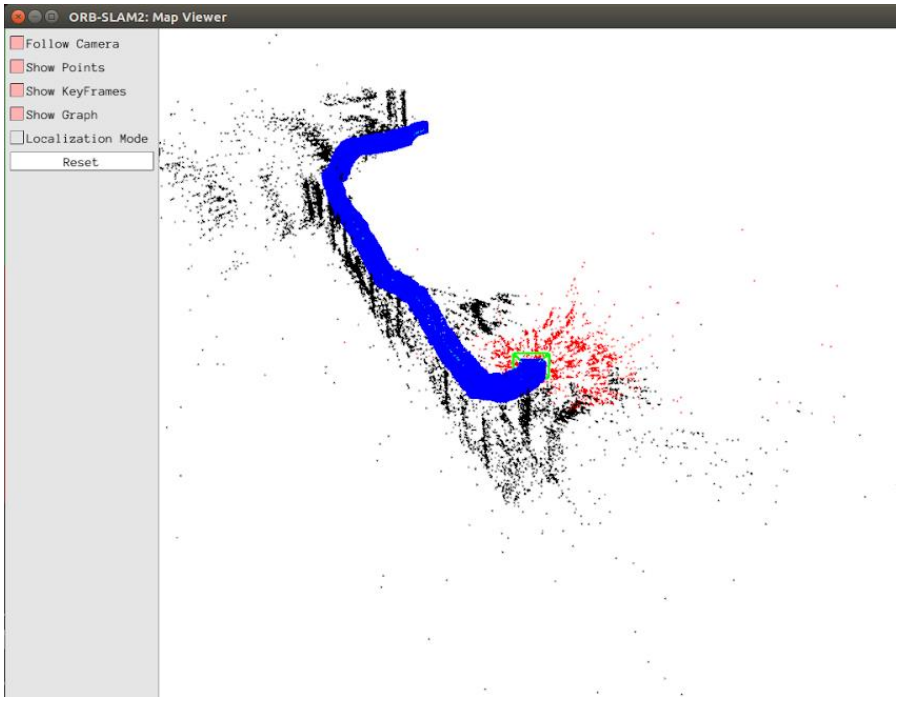


Figure 3.8 ORB-SLAM2 sparse point cloud map viewer based on Pangolin. The blue line (overlapping blue squares) are the trajectory of the camera. The red points are current feature points in the camera.

However, we have to generate a dense point cloud map in the system to allow it to detect and locate a tree position. A directly generated point cloud method is used in this research based on the depth image from the camera and estimated camera position. There are three reasons why we do not use sparse point clouds to generate dense point clouds.

- The sparse map is too sparse to allow the dense map to be generated with enough true information.
- Computational resource consumption is not smaller than the direct generation of dense map.
- The amount of generated 3D map from sparse map information is less than directed generated map from the RGB-D sensor.

Through the RGB-D sensor, a real-time point cloud can be generated directly. However, the cumulative errors of translation matrix from camera pose estimation may cause the global point cloud map to have many layered and staggered phenomenon. Therefore, in order to generate a real dense point cloud, the following four subsections outline a process of the 3D point cloud reconstruction map, including data filter, registration, reduction, and model reconstruction.

3.2.3.1 Point Cloud Filter

After point cloud acquisition by an RGB-D camera, it is necessary to filter the point cloud according to the actual problem. Similar to 2D image filter, there are several situations needed to filter the point cloud:

- There are outliers in the point cloud.
- Point cloud clustering contains a large number of points, resulting in excessive calculation.
- The cloud contains noise data which needs to be removed.

However, the difference between the 3D point cloud filter with the 2D image (color and depth) filter is that the 3D filter needs to consider the depth value noises and outlier points. Moreover, the 3D point cloud map has much higher data calculation than a 2D image filter. The computational resources must be reasonably allocated so that the entire detection system has the ability to detect a tree in real time.

Three filter algorithms are used to filter the outliers in our collected point cloud, pass-through filter [Rusu in 2011], voxel grid filter [Rusu in 2011] and statistical outlier removal filter [Rusu in 2011]. Considering the available view range of the RealSense R200 camera in an outdoor environment, we set six meters as the pass-through threshold that means we will not consider points further than six meters away from the camera as shown in **Figure 4.2 and 4.3** in **Chapter 4**.

When using the high-resolution camera to collect point cloud, it is too dense. An excessive number of cloud points cause difficulties in subsequent segmentation. The voxel grid filter can achieve the function of down-sampling while not destroying the geometric structure of the point cloud. The point cloud geometric structure is not only a macroscopic geometry but also has micro-arrangement, such as the similarity of the horizontal and vertical size. It is instrumental in improving the algorithm speed of registration, surface reconstruction, and shape recognition. In detail, the first step is to

create a 3D box as a voxel grid from the input point cloud data. Then for each box, the centroid of all points in the voxel is used to show the other points in the voxel. All the points within the voxel are finally represented by only one centroid for an entire voxel. In addition, random sampling has higher efficiency than the voxel filter.

A statistical outlier removal filter is used to remove significant outliers from the measurement noise. The feature of outliners is sparsely distributed in space. Each point expresses a certain amount of information. The greater density of a region has the greater amount of information. Noise information is useless information; the amount of information is small. The information expressed by the outlier is negligible. Considering the characteristics of the outlier, if the density of point cloud of a particular area is less than a threshold, the points in the area can be removed.

There is a gap in the automation and measurement accuracy between point cloud filtering with image filtering and signal filter. In fact, the low-level point cloud pre-processing has a significant impact on the high-level accuracy and speed of detection, recognition, and localization.

3.2.3.2 Point Cloud Registration

The problem of aligning multiple 3D point cloud views together into a complete model is called registration [Weinmann et al. 2015]. There are two steps to achieve the registration process. The first step is to align the point clouds in different coordinate systems to the same coordinate system. Then a set of coordinate transformation parameters is optimized by ICP iteration to minimize the registration error. The purpose of the first step is to determine an initial coordinate transformation. The point cloud of different coordinate systems is roughly unified into a global coordinate system. In essence, this step is to use the camera transformation matrix to do the transformation process for each frame of the point cloud. Previous motion estimation systems generate the camera transformation matrix.

In order to satisfy the accuracy requirement, step two must be used to reduce the registration error between each two of point clouds. By defining an error function to reflect the degree of coincidence between the overlapping regions of the point cloud, a set of coordinate transformation parameters is optimized based on the iterative closest point (ICP) algorithm. Therefore, the defined error function is minimized.

The objective of the ICP algorithm is to solve the transformation between two heap points cloud datasets. From the candidate translation matrix of camera pose, the rotation (R) and

the translation (T) can be determined. If there are two sets of point clouds X and Y with m and n points, the solution of the problem can be described as minimizing the mean square error:

$$\mathrm{E}(\mathrm{X},\mathrm{Y}) = \sum_{i=1}^{m}(R_{x_i} + T - y_i)^2 \quad (11)$$

The ICP optimization algorithm needs to initialize the R and T through estimation between camera poses, and minimize error matrix. Moreover, the structure of the points need to be optimized by Levenberg-Marquardt with different kernels or SVD algorithm. For each point in X, use the current R and T to find the nearest corresponding point in Y, these two points in X and Y can be seen as a pair. Finally, every pair of points in the two point clouds will be used to iterate the rotation (R), and translation (T) to converge the minimum mean square error E. ICP algorithm is simple, intuitive, easy to operate and has a good registration accuracy. However, there are erroneous corresponding points in the corresponding points established by the above method. For this reason, the algorithm needs to introduce evaluation criteria or constraints to remove the error corresponding point pairs.

3.2.3.3 Data Reduction

Considering the goal of this research, the final detection result needs to first calculate a 360° of point cloud data within six meters. Although previously collected point cloud data has been real-time filtered, the final output of the global 3D map still has a huge number of points. Excessive data points can lead to lower operating efficiency of the computer and storage space increases. In addition, the tree detection algorithm also needs to consume more time. Therefore, a global filter, similar to the previous real-time point cloud filter, can be used to filter the global registered 3D point cloud map.

3.2.4 Loop Closure Detection

Loop closure detection is essentially an algorithm for detecting the similarity of observed data. In V-SLAM, Bag-of-Words (BoW) method is used by most of the algorithm, RGB-D SLAM, RTAB SLAM, and ORB SLAM. The BoW model classifies the visual features (SIFT, SURF, ORB, etc.) in the image, and then creates a dictionary to find the features. In the research, the loop closure detection selects the ORB SLAM loop detection method. The RGB-D ORB SLAM loop closure detection method has two steps, candidate loop detection and validation, and pose graph optimization.

ORB SLAM uses DBoW2 as the first part of the loop closure detection algorithm to detect the loop candidates. The establishment of BoW dictionary in DBoW2 has four steps:

- It extracts features from the image.
- The k-means algorithm clusters the extracted features and the description subspace is divided into k classes.
- Each clustered subspace continues to be clustered by the k-means algorithm.
- To iterate above three steps, the descriptor is built into a tree structure.

In addition, [Gálvez-López et al. 2012] mentioned that when the dictionary tree needs to be inserted into a new image, the extracted feature descriptor starts from the root node of the dictionary tree step down to the leaf node based on the Hamming distance. The frequency of each image with similar features can be calculated. A high frequency indicates the degree of discrimination of the image feature is small.

The similarity transformation computation is only for monocular SLAM to drift scales. In this research, the RGB-D camera can directly observe the real scale based on depth information. Therefore, the pose-graph can be generated directly by the RGB-D sensor without correcting the scale drift. After the pose-graph obtained from loop closure detection, a full bundle adjustment (BA) is used to optimize the global point cloud map to get the best solution. The detail of BA is shown in **Appendix A**.

3.3 Tree Detection

In this section, the optimized global point cloud map from the above method is used to achieve our research purposes, tree detection, and localization. Our tree detection system consists of two parts. The first part is estimating the relative and approximate position of surrounding trees by analyzing the previously reconstructed point cloud map. According to the camera current pose and the position of the trees, the relative closest tree can be calculated. The second part is when the nearest tree position is known, if the tree is in the RGB-D camera field of view, the exact location and the radius of the tree is measured in real-time through a 2D depth image.

3.3.1 Tree Detection in 3D Point Cloud

In general, a direct detection method based on 3D geometric model surface fitting will identify some simple models in a point cloud map, such as a plane, cylinder, circle, etc. There is a common object detection method that directly segments the fitted 3D objects

from the global point cloud map. According to its use of mathematical methods, there are several existing object segmentation algorithms based on point cloud as mentioned in 19.

[Rusu in 2011] claimed that the classical random sampling consistency segmentation (RANSAC) could only segment a model from a specific point cloud data set and does not apply to the point cloud segmentation problem with multiple point cloud clustering. The regional growth segmentation can segment the cloud clustering well, but the time complexity of the algorithm is enormous and does not apply to the real-time scene. Although the Euclidean cluster segmentation is fast, it can easily cause insufficient or excessive segmentation. The min-cut segmentation has a high segmentation accuracy, but the segmentation process is not completely automatic. The locally convex connected patches method combines with super-voxel and the regional growth algorithm to obtain a good performance of point cloud segmentation, but there is still a serious excessive segmentation.

These segmentation methods have some good performance in some specific environments, however, in our research environment, these methods are not suitable. Therefore, we design a novel detection method by reducing the dimension from the slice of the 2D point cloud to fit a circle model in certain axis (y-axis presents height in this case). These fitted 2D circles are used to estimate their shape in 3D space. The method reduces the complexity of the tree detection in point cloud map. It uses sliced 2D model points to fit a half circle for each sliced cluster point. In this research, we propose a point cloud slicing algorithm to detect and localize tree candidates as a cylinder in point cloud map. In detail, the method has four steps:

3.3.1.1 Slicing

The point cloud is a collection of scattered points in space that characterize the surface profile of the objects in the environment. Point cloud slicing can be described as follows: For a given direction of its tangent as the normal vector to construct a plane cluster. Two important parameters need to be determined in the slicing algorithm, slice direction, and slice thickness. In this case, the slice direction is based on the y-axis (vertical/height). We set the initial camera pose at the origin and its faces to the direction of the z-axis. In the real world, the initial position of the camera is horizontal and close to the ground in any direction. In addition, a plane model fitting algorithm is used to estimate the specific location of the ground within a limited height region. Therefore, the slice direction is perpendicular to the plane. In theory, the direction is the same as y-axis vector.

Because of the point cloud discrete expression restrictions, it is expensive to calculate the outline of the object strictly based on the sliced plane point cluster of the point cloud. Therefore, we need to have a reliable estimate of the slice thickness (*t*) that is the distance between the two slices. If the slice thickness is too great, the 2D sliced contour curve error is too high, and the calculation efficiency is low. If the slice thickness is too small, the 2D contour is distributed as a multi-segment disconnected curve. The global point cloud map has a down-sample filter process before reconstruction. The density of the point cloud map is averaged. We can refer to the average density (ρ) of the point cloud to predict the thickness of the slice. The density ρ is expressed below:

$$\rho = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} D_i^j}{n \times m} \tag{12}$$

Where a set of point (n) is randomly selected from the point cloud, for each point (g_i) in the set, the closest points (m) of the set of point (n) are found through kd-tree by a certain threshold. Then, the distance of each of m point can be found and we can figure out the density (ρ) of the point cloud from the average of distance. From the density of point cloud, slice thickness (t) can be estimated by a constant parameter k, as shown in below equation.

$$t = k\rho \tag{13}$$

After the slice thickness has been determined, the next step is projecting each sliced 3D point cloud to 2D that means find the intersection between the slice plane (S_i) with point cloud. For each slice, two set of points are determined by the slice and the distance of half thickness (-t/2 & t/2) as upper points and lower points. For each of point in lower points, the closest point is found by a kd-tree algorithm from upper points and so we figure out the distance. If the distance is smaller than a certain threshold, the intersection points of the two points ($p_u i$ and $p_l j$) in the plane can be calculated. The **Figure 3.9** shows the project method from point cloud to a 2D slice plane.

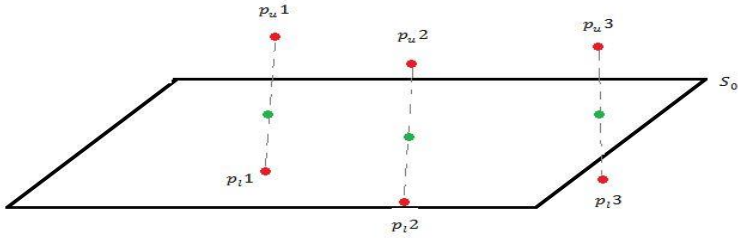


Figure 3.9 The project method schematic diagram. Six 3D points (red) are projected to the slice plane $\boldsymbol{S_0}$ as three intersection points (green).

3.3.1.2 2D points Clustering

This step clusters the set of points for each sliced segmentation in 2D. Compared with the clustering algorithms in 3D, the 2D clustering algorithm is cheaper. In the research, the Euclidean clustering algorithm proposed by [Kiranyaz et al. 2011] is used to cluster the trees in the 2D sliced points. Euclidean clustering is a special Kernel k-means clustering algorithm; it can explicitly map real data to the same dimension of the complex space. The Euclidean clustering algorithm uses the Euclidean distance equation (as shown in below) to determine the center of clusters.

$$d_{x,y} = \sqrt{\sum_{k=1}^{n}(x_i - y_i)^2} \tag{14}$$

We can cluster the tree trunk and crown as circles in the 2D sliced point set, as shown in **Figure 3.10**. In addition, the clustering method can also reduce the consumption of computing resources. Some clusters with only a few points can be ignored.

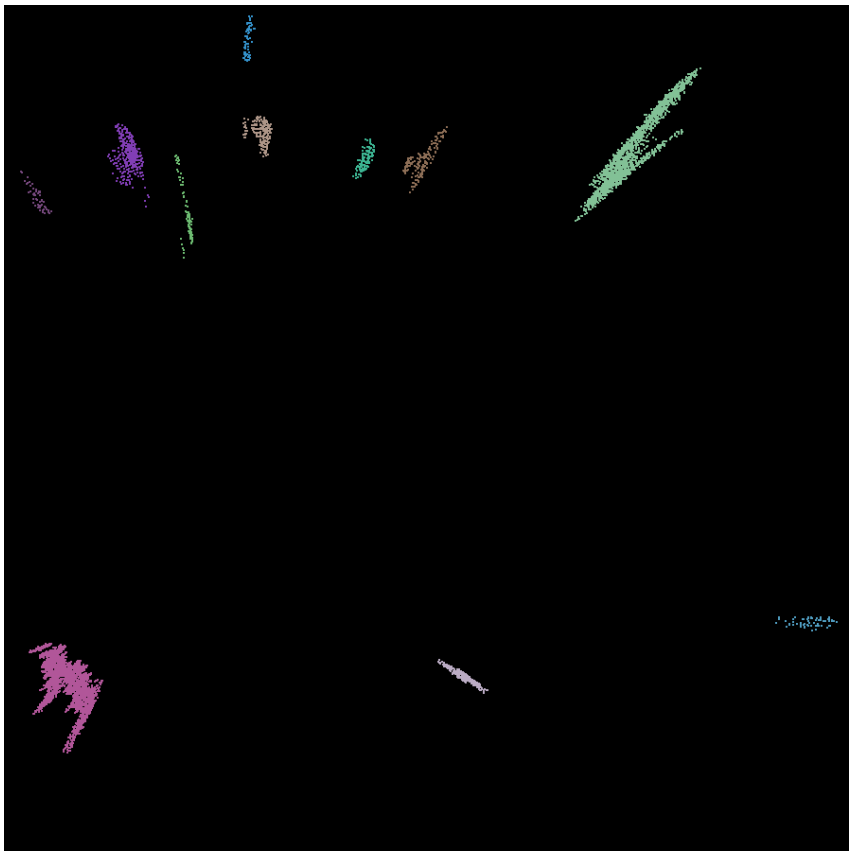


Figure 3.10 The figure shows the 2D sliced points clustering for one of the sliced point clouds. The different color points mean the different clusters.

3.3.1.3 Fitting Circles in 2D

To fit the circles in each 2D slice points is a significant step to finding trees in 3D space. The process of fitting in this research is divided into two stages. A 2D convex hull

algorithm proposed by [Susan et al. 2017] is the first step to fit circles into the 2D sliced points. In order to reduce the complexity of the fitting algorithm, the edge set of each 2D cluster is calculated by the 2D point cloud convex hull algorithm. The set of points can be connected to form a minimum polygon containing all the cluster points. The second step uses a 2D circle fitting algorithm which is based on the non-linear least squares algorithm, Levenberg-Marquardt method, provided by [Chernov et al. 2010]. The fitting algorithm can refer only the extracted convex hull points to determine a circle. The method significantly improves the circle fitting speed.

The convex hull algorithm in the research uses the Graham scanning method [Susan et al. 2017]. The basic idea is to solve the convex hull problem by setting a stack at the candidate point. For each clustering point, this method can only preserve the set of convex hulls for fitting a circle. In the research, we found that using the convex full algorithm to extract the hull point set from a set of cluster point can speed up the circle fitting algorithm. However, this method can only be used to detect an approximate contour of an object. Some objects with concave characteristics are not recognized. Through the above method, each cluster of each layer is fitted by a different size circle. We can also assume that each of these points is a collection of circles.

3.3.1.4 Fitting Cylinder in 3D

The accuracy of tree detection depends on the restrictions of the physical parameters of the tree. The assumptions about the geometric restrictions of the tree are shown below.

- Grown vertically, which means regardless of the slope of the ground, the trees grow vertically in the reverse direction of gravity.
- Radius restrictions, which means trees are limited in size. 3cm – 20cm radius in the research.
- Single trunk, which means there is no obvious bifurcation of trees within one meter above the ground.

Because there is only the point cloud model of a tree, the detection is difficult to achieve through the texture and color. Therefore, the surface geometric parameter of the 3D tree trunk model can only be predicted via the above three assumptions. According to above, the cylinder fitting based on the previously detected circles can divide into two steps.

- Ignore all circles whose radius exceeds the threshold.

- Determine whether each sliced layer has circles with similar radius in similar 2D position.

If there are enough sliced layers with similar circles from the ground to the top of the point cloud, the detected circles can form a cylinder. The detected cylinder can be seen as a candidate tree in the point cloud map as shown in **Figure 3.11**. The detected tree candidates are presented as white cylinders for three different camera moving methods. The top is the single frame of the camera face to one tree. The middle point cloud represents a row of trees on the side of the street. The bottom map shows all trees around.

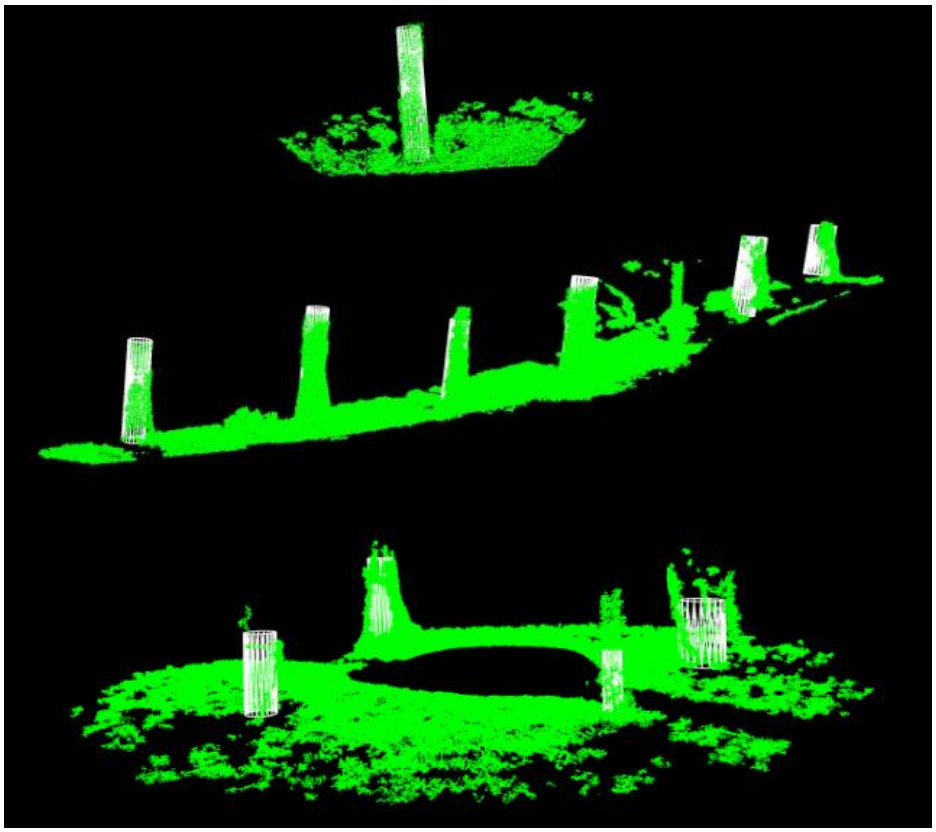


Figure 3.11 Tree detection visualization in the 3D point cloud.

These detected trees (cylinders) can be stored as a class with only the approximate coordinates and radius. As for its point cloud, data can be stored and stop the calculation because the position of the surrounding trees has been obtained. According to the tree positions and the current UAV pose, the closest tree relative to the camera can be determined. The system can control the drone face to the nearest tree through publishing an ROS topic to the flight control.

3.3.2 Tree Detection in 2D Image

Due to the limitations of the RGBD camera field of view, the camera has to be rotated 360° to get a surrounding point cloud map. However, during the rotation of the camera, V-SLAM will cause the cumulative error to affect the exact location of the trees. In our research experiment, the error of the detected tree center coordination has 0 to 0.8m. The error of the detected tree radius has 0 to 0.07m. This error cannot be ignored when a UAV needs to navigate to a tree. In order to achieve more accurate navigation, when the UAV camera turns to the target tree, an exact tree location is re-estimated through the depth image and previous point cloud detection results.

3.3.2.1 Object Detection in Depth Image

In addition, location of a tree can be determined in a 2D image as shown as a red rectangle in below **Figure 3.12** and the radius of the tree can also be displayed in the 2D image. Moreover, the projected tree radius can also be changed with the modification of the distance between the tree and the camera. However, there is a certain deviation between the projected tree position and the ground truth tree position in the 2D image, as shown in **Figure 3.13** This deviation has had a significant impact on UAV navigation. Therefore, this section will describe a method to adjust the position of the tree using the depth image.

Figure 3.13 An estimated tree position in 2D RGB-D camera image from a 3D tree location. The ground truth is the image of the tree, and the mapped tree is shown as a red rectangle. The position of the tree is measured without loop closure detection.

After obtaining the approximate position of the tree, an exact tree position can be detected and located by a depth map or a color map. Currently, deep learning is the most popular method used to detect objects in 2D images. There are three key benefits of image deep learning, location, recognition and classification. In this case, only the location of the objects is required by the system. Although these two algorithms can get a good result, the consumption of the computing resource is quite large for deep learning and it is difficult to determine the depth information of the objects. Therefore, we have not applied deep learning to detect the tree position in the 2D images.

The depth image is equivalent to replacing the depth information with color information on each pixel of each frame of the color image. Based on this principle, the ground plane and the objects on the ground can be obtained in each frame. Therefore, the first step is to identify the ground plane. Normally, if the pixel is on the same plane, the trend of its depth change is slow. When the object on the ground, the depth value of the corresponding pixel changes significantly. In the research, the algorithm traverses the depth values of each row of pixels on the image from bottom to top. Excluding situations where the depth value is too large, through previous depth image filtering, the depth value only includes zero and 0.3 meters to 6 meters. We ignore pixels with a depth value of zero and only consider the objects above the ground.

For objects above the ground, we binarize the depth image and extract its contours. For contours, the different depth value of the farthest point and the closest point must not exceed a certain threshold, 0.5 meters in the research. Moreover, we then calculate the convex hull for each contour and filter out the smaller convex hulls. In these larger area convex hulls, if they are in the vicinity of the 3D mapping position of the previously detected tree, and satisfy the constraints of a tree, suitable radius, and single trunk, the filtered convex hull can be seen as a tree in the 2D image.

3.3.2.2 Kalman Filter Tree Tracking

However, the tree detection result has quite significant noise due to the sensor error. In order to filter out these interference signals, we use Kalman filter to remove the noise and estimate a reliable tree position in the 2D image. The Kalman filter is a linear recursive filter that optimizes the next state based on the previous state sequence of the system. In the research, we use two one-dimensional Kalman filters to track the tree position and radius in the 2D image, because these two tracking values are independent. In general, the algorithm of the Kalman filter is basically as follows:

1) Set the mean and variance of the initial state, x_0, P_0, when t = 0.
2) Accept the next observation, v_t
3) Calculate the variance P, R, and the Kalman gain K.
4) Update the estimated variance and mean, x_t, P_t
5) Back to step 2, t + 1.

Usually, the camera has three kinds of motion trajectories relative to the tree. Therefore, there are three different Kalman filter tracking experiments to test the projected tree position and radius in the 2D image.

- Go left and right
- Go ahead and backward
- Random moving

To simplify the model, we assume that the camera object tracking system is a linear discrete system. The state of the UAV is $x_1$to x_k where k is the discrete time subscript. The state equation and observation equation of the camera observation system can be respectively described as:

$$x_k = f(x_{k-1}, u_k, w_k) \tag{15}$$

and

$$z_k = h(x_k, v_k) \tag{16}$$

For the state equation, where $f()$ is motion equation, u is the input value, and w is the noise of the input value. For the observation equation, where $h()$ is the observation equation, z_k is the observed data, and v_k is the observed noise. The equation of motion describes how the state x_{k-1} changes to x_k, and the observational equation describes how x_k gets the observed data z_k.

Kalman filter is the unbiased optimal estimate of the recursive form of the linear system. The conclusion of the Kalman filter equation can be represented as three parts:

- Prediction value and noise

$$\tilde{P}_k = A_{k-1}\hat{P}_{k-1}A_{k-1}^T + Q_k \tag{17}$$

and

$$\tilde{x}_k = A_{k-1}\hat{x}_{k-1} + v_k \tag{18}$$

For the value equation, A is the translation matrix, P is the prediction, and Q_k is the covariance matrix of the noise.

- Kalman Gain

$$K_k = \tilde{P}_k C_k^T (C_k \tilde{P}_k C_k^T + R_k)^{-1} \quad (19)$$

The Kalman gain K reflects the confidence of the measurement results with the process model. Where C is the observation matrix, R is a covariance matrix to indicate the uncertainty in the observation.

- Update adjustment value and noise

$$\hat{P}_k = (I - K_k C_k)\, \tilde{P}_k \quad (20)$$

and

$$\hat{x}_k = \tilde{x}_k + K_k (y_k - C_k \tilde{x}_k) \quad (21)$$

The above two equations update the adjusted state and noise distribution of the optimal estimate. As mentioned before, $\tilde{x}_k$ is predicted based on the previous state. The difference between it and the optimal estimate is the Kalman gain $K_k(y_k - C_k\tilde{x}_k)$. $y_k - C_k\tilde{x}_k$ represents the residual between the actual observed value and the estimated observed value. These above five equations can be used to correct the predicted tree positions based on the observed positions to achieve the purpose of optimal tree position estimation.

3.4 Navigation in ROS

In the research, a medium-sized consumer UAV is used for our tree branch pruning project. To allow the UAV to fly in a stable posture, and efficiently and intelligently complete some missions in a complex environment, we require a reliable flight control system to control the drone. We load an open source flight controller, Pixhawk [Gültekin et al. in 2016], on our real UAVs. Pixhawk is an ARM-based 32-bit open source flight control, which was developed by ETH's computer vision and geometry group [Meier et al. in 2015].

In order to avoid damage to the UAV, a simulator, Gazebo, is used to simulate the flight controller PX4 SITL to test the tree detection system. The simulator executes an entire flight control system on the host. The PX4 SITL is connected to the Gazebo 7 simulator through the MAVLink based on ROS. The interface of the MAVLink of ROS is MAVROS. It is built on the topic and server to communicate a message between each. In the real system, a separate MAVROS thread that periodically sends instructions is

necessary. The setup of the system looks like this provided by [Meier et al. 2017] **Figure 3.14**:

Figure 3.14 The setup of UAV flight controller simulator Gazebo 7 based on ROS.

In the research, our purpose is to consider the reliability of the V-SLAM. Therefore we do not use the GPS in the simulator, a V-SLAM-based visual estimation system is used to build the flight control system. The V-SLAM is used for our docking navigation between the UAV and the target closest tree.

3.4.1 UAV Movement Process

We divide the UAV movement process into four steps in the research, include:

- Take off
- Rotation (mapping)
- Rotation (find target)
- Docking (flying to the tree and hovering near it)

As shown in **Figure 3.15**, the initial position of the UAV is on level ground according to the departure requirements. The camera position is probably 10 cm higher than the ground. Then the UAV slowly rises to 1.5 meters from the ground and is suspended in the air. The second step is controlling UAV rotation 360 ° at the origin by the flight controller to obtain the surrounding color and depth image information. The 3D point cloud map is reconstructed through the visual SLAM based on the image information. After the point cloud map detects the rough tree position through our slicing method, the closest tree can be determined. The next step controls the UAV to rotate a certain angle to ensure the closest tree is in the field of view of the camera. The final step of the research allows the UAV to move toward from a few meters away to the tree (about one meter away from the tree).

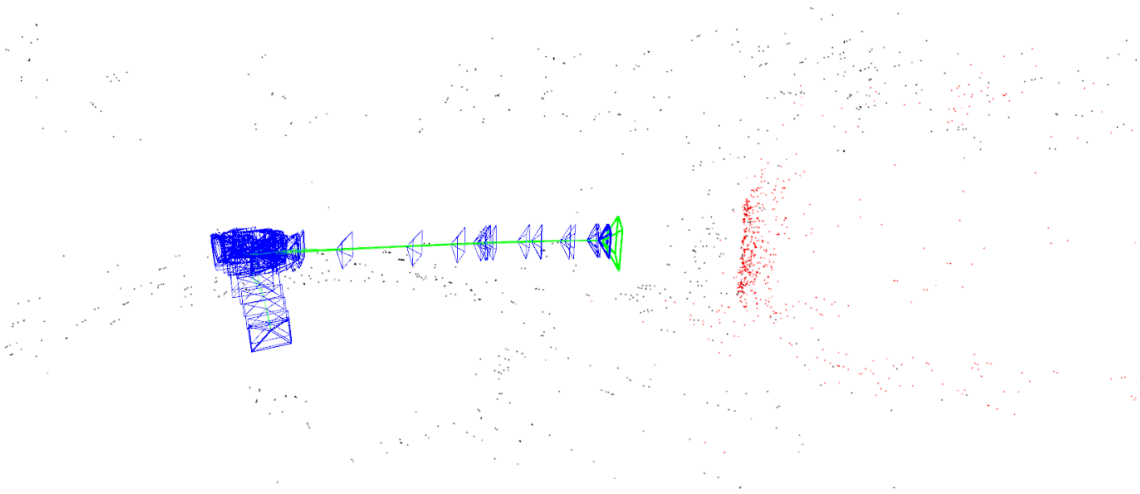


Figure 3.15 The handheld camera tree detection navigation simulation trajectory.

3.4.2 Handheld Camera Simulation

Because the obtained depth of the information shows almost no error on the simulator, in order to ensure that Intel® RealSense™ Camera R200 can also provide reliable data for the algorithm in the outdoor environment, we carry out a simple experiment. We use a handheld R200 camera to send a UAV's direction control command to the experimenter while acquiring the data from the camera. There are six commands including, take off, landing, turn left, turn right, toward, backward and stop. The experimenter moves the camera according to the obtained command.

Therefore, we simulate the real UAV movement situation. Firstly, we position the camera 10cm above the ground while running the proposed tree detection algorithm. When the SLAM starts tracking the first keyframe and gets the initial camera position, the experimenter slowly raises the camera to a position 1.5 meters above the ground. Secondly, while maintaining the current height, the experimenter turns the camera 360° to build the surrounding point cloud map. The previous two steps are a constant UAV operation. The purpose of these two stages is to create a point cloud map and find the relative position between the nearest target tree and the camera.

According to the obtained relative position, the next step controls the camera face to the tree. Because the depth image field of view of the Intel® RealSense™ Camera R200 is vertical (46°+/-5°) and horizontal (59° +/-5°) [Intel RealSense Datasheet], if the system wants to know the exact location of the tree, the tree must be in the field of view of the camera. Therefore, the UAV needs to control the direction to allow the closest target tree into the field of view of its camera through rotating an angle. According to the previous

detection result, the approximate closest tree position is estimated by the point cloud model, and the V-SLAM algorithm has also estimated the camera pose. Through these following three known conditions, the system can directly map the tree into the 2D depth image.

- Field of view.
- Target Tree 3D coordination, T.
- Camera 6 DoF pose, C_p.

In addition, when the target tree is not in the field of view, the system can control the UAV to capture the tree through the minimum rotation angle. For example, through the current camera pose matrix C_p, the direction of the camera is known. The approximate tree position T is also detected already. Therefore, an angle θ between the tree and the camera with the camera direction can be calculated. In the example, a turn right θ degrees ROS topic is published to the UAV flight controller, as shown in **Figure 3.16**.

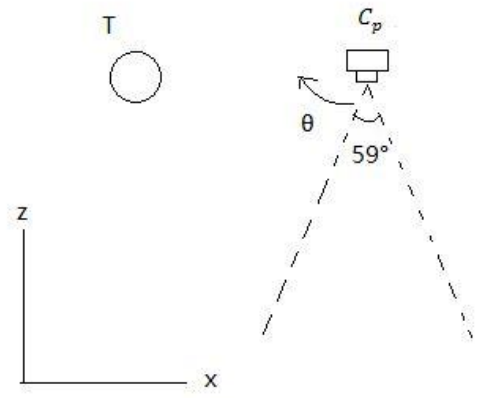


Figure 3.16 Top-down view: the rotation control of the camera when the closest tree is not in the field of view.

Simply, the control instruction of this step is only to turn left and turn right. Because the UAV is assumed to hoving stably in space where it is 1.5m above the ground, the step can be seen as being done in a 2D flat space. After the closest tree is in the field of view of the camera, the camera is moved to a position one meter from the tree following navigation commands. The navigation of this step is achieved through detecting the tree position in the 2D depth image. There are six commands:

- Turn left
- Turn right

- Toward
- Backward
- Stop
- Docked

As shown in **Figure 4.15** and **Figure 4.16 (Chapter 4, section 4.2.1)**, When the camera moves to the specified position, and the camera stops moving, this appears as a green border on the colour image. The above steps are all processes of the research.

3.5 Chapter Summary

This chapter introduces our proposed method for the tree trunk detection and navigation in a forest environment based on a small-medium size UAV with RGB-D camera. Firstly, we detail the hardware, including the Intel® RealSense™ Camera R200. The operating system and the development environment is also detailed. In addition, the pre-processing of the depth map is also described, such as denoising, hole filling and outlier removing. The ORBSLAM2 is applied to locate the camera pose via visual SLAM algorithm in the research. Furthermore, the density 3D point cloud map is built through the localization of the camera pose to detect rough tree positions. According to the approximate tree position, we can further locate the more accurate position through the depth map directly from the camera. These algorithms are written in C++, most of them are based on PCL, OpenCV, and CGAL. Finally, a UAV flight control simulator system based on PX4 and Gazebo, which can be controlled by the ROS topic and server, is described in the last section to perform the experiment of the navigation of the proposed method. In addition, we also use the handheld camera to test the performance and accuracy of our proposed tree detection algorithm in a real outdoor environment.

4 RESULT AND DISCUSSION

This chapter discusses and analyzes the performance and evaluation of the experiment results of our proposed methods presented in **Chapter 3**. This chapter shows the tree detection and navigation experiment results. These results are analyzed and we discuss the advantages and disadvantages of our proposed method.

4.1 Tree Detection

4.1.1 Depth Threshold

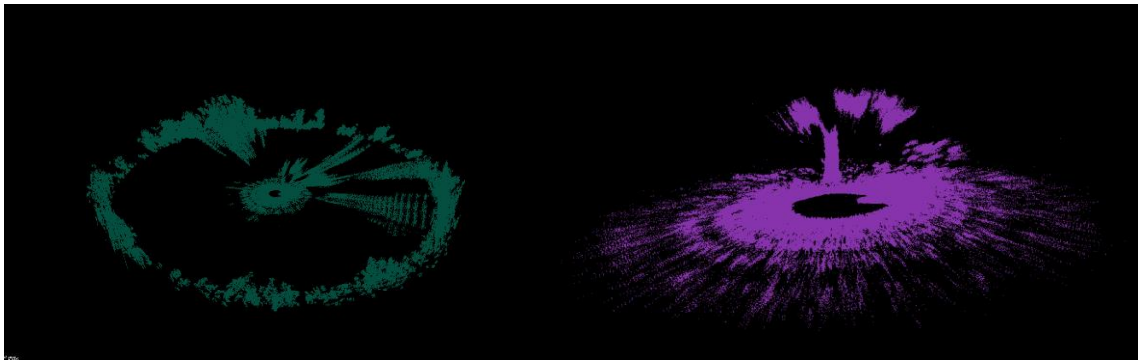
(a) (b)

Figure 4.1 (a) The raw depth registers an image from R200 via ROS Linux driver. (b) The depth image in the range 0.5 meters to six meters.

According to our experiment and [camera product datasheet], the effective measurement distance of the R200 camera is greater than 0.5 meters and less than six meters. The depth threshold removes the depth data beyond this range. As shown in **Figure 4.1**, the raw depth image (a) contains a lot of noisy background redundant data. In **Figure 4.1**. (b), the background redundant data and some close noise values are removed as well. As shown in **Figure 4.1**. (b), part of background redundant data is replaced in the raw depth image.

Through the removed depth map, the point cloud map which is generated from the depth map has less background redundant data. As shown in **Figure 4.2** (a), the point cloud map is generated by the raw unthresholded depth image. Because the far points are noisy, it cannot be used for the next point cloud processing. The **Figure 4.2** (b) shows the point

cloud map after removing the useless background data. This redundant data not only affects the accuracy of the algorithm but also increases the amount of computing time. In fact, the raw point cloud without thresholding has 1,856,481 points. After the threshold filter, the number of points in the range of 0.5 meters to six meters is only 1,073,888. In addition, the average traversal time of the cropped point cloud map (6302 milliseconds) has only 0.59 times of the raw map (10888 milliseconds).



(a) (b)

Figure 4.2 (a) The raw point cloud generated via the raw depth image without range threshold includes 1,856,481 points. (b) The raw point cloud after the 0.5m to 6m threshold cut includes 1,073,888 points.

4.1.2 Data Filter

However, the approximate six seconds point iteration time is too long for the research, so we use the two filters, down sampling filter and statistical outlier removal filter as mentioned before, to reduce the traversal time from 6302 milliseconds to 1373 milliseconds. As shown in **Figure 4.3** (top), the down sampling filter significantly reduces the number of points in the raw point cloud **Figure 4.2** (b) from 1,073,888 to 238,432 for the 0.01 down sampling leaf size.

The **Figure 4.3** (bottom) shows the point cloud map after the statistical outlier removal filter. Many points in the air and the edge of ground in the point cloud are removed. Although the outlier removal only reduces the number of points from 238,432 to 216,219, it can significantly mitigate the tree detection false positive impact from the outlier points. For the step, the traversal time of the point cloud map after these three filters above is approximately ten times faster than the raw point cloud map without filters.

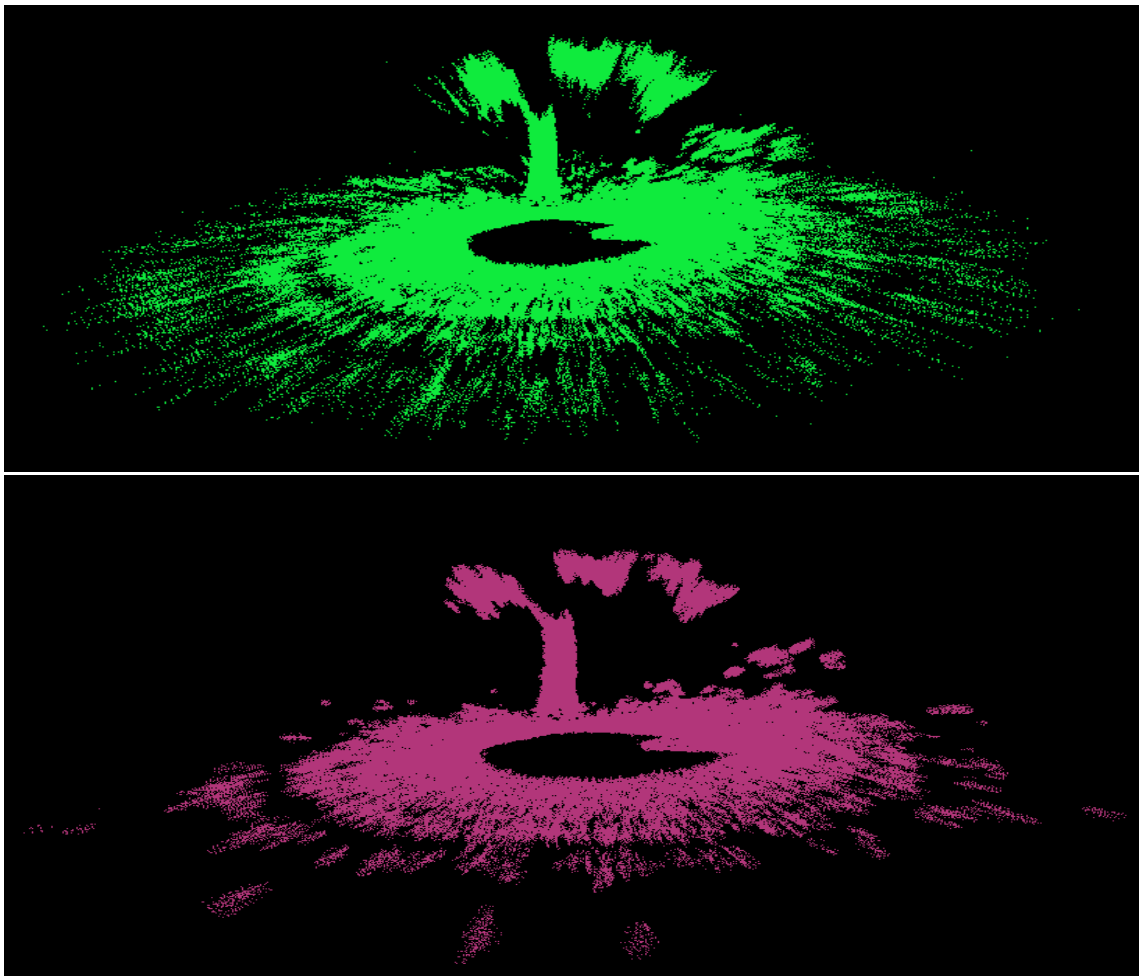
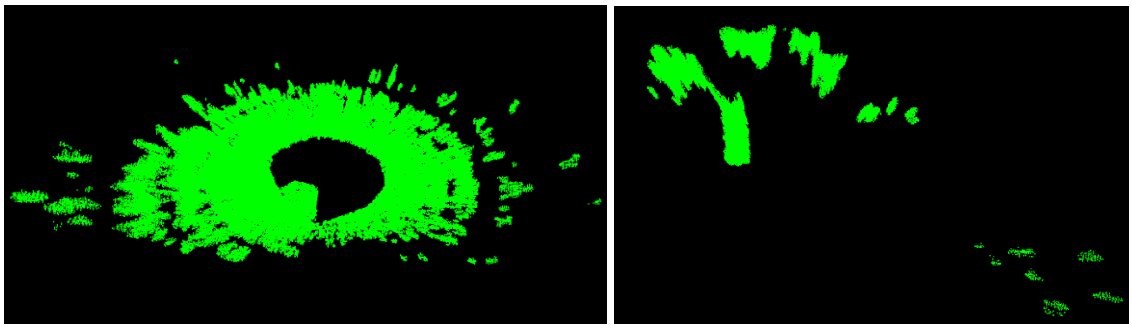


Figure 4.3 (top) The point cloud map through the downsampling filter includes 238,432 points. (bottom) The point cloud map through the statistical outlier removal filter includes 216,219 points.

4.1.3 Ground Removal

These above filters are normal point cloud pre-processing methods. They can be used in most of the point cloud map processing. However, in the research, the points which represent the ground are not useful for the tree detection. We need to filter these redundant points. As shown in **Figure 4.4**, (a) image is the extracted ground points via ground plane detection algorithm and passes through the filter. In this case, the ground has 171,095 points, and the **Figure 4.4** (b) image shows a point cloud map with 45,124 points as objects above the ground. The point cloud map after the ground extraction is the final point cloud before executing the 3D slicing tree detection algorithm. In general, the point of the ground occupies half, or even two-thirds of the whole point cloud map, depending on the number of objects above the ground. Through the above algorithms, the traversal time of the point cloud can be compressed within less than one second without GPU

accelerator. The next subsection describes the execute time and results analyzing of the specific 3D tree detection based on point clouds.



(a) (b)

Figure 4.4 (a) The ground points extracted from outlier removed cloud map. (b) The tree and other objects point above the ground after the ground removal algorithm.

4.1.4 Point Cloud Detection

4.1.4.1 Point Cloud Slicing

The first step is slicing the point cloud from the ground plane up to the top of the point cloud. As mentioned before, the slice gap is determined by the resolution of the filtered point cloud map via the RANSC method by the equation (12) that is the point cloud density equation. In addition, because the RANSC method and the voxel down sampling points are not an average distance, the measured densities of the point cloud have a small difference. In fact, the difference will not affect the slicing results and can be ignored. The average resolution of the filtered point cloud maps that has 0.01 voxel leaf size is 6.91mm. We compare the different voxel leaf size in the research to determine the density of point cloud as shown in **Table 4.1.** We finally set the leaf size as 0.01. Because the execution time of the point cloud detection algorithm is within 10 seconds, the slicing thickness is determined by the density and a constant number. We set the constant number as 4, which means the thickness is four times the density as shown in equation (13).

Table 4.1 The comparison of density and slicing thickness to compare different voxel leaf sizes. Units are in meters except running time.

Voxel Leaf Size	**Density**	**Slicing Thickness**	**Running Time**
0.001	0.0022	0.0088	64.4 sec
0.05	0.0046	0.0184	34.7 sec

0.01	0.0069	0.0276	8.9 sec
0.02	0.0125	0.05	5.1 sec
0.03	0.0203	0.0812	3.0 sec

In addition, according to the measurement distance of the camera, when the distance of the tree to the camera is within three to five meters, the trunk of the tree (approximately two meters) can be sliced into six to twelve layers. The number of layers in the range can help the algorithm determine an object. As shown in **Figure 4.5**, a tree trunk is sliced into nine layers, and a low bush near by the tree is divided into only four layers.

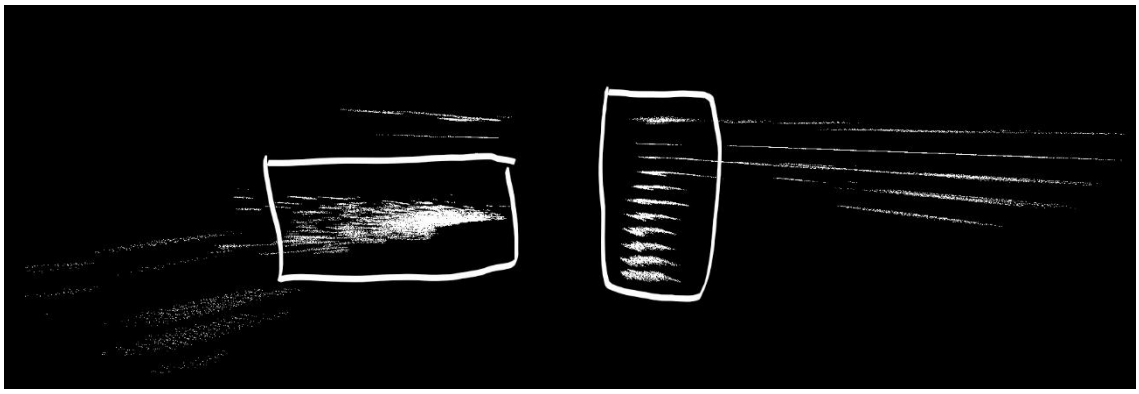


Figure 4.5 The sliced point cloud map. In this case, the down sampling voxel leaf size is 0.01. The slicing thickness is 0.0281 meters. A tree trunk is sliced into nine layers. Each layer has a similar center and radius.

4.1.4.2 Euclidean Clustering Sliced 2D Point Cloud

For each slice of the point cloud, we use a Euclidean clustering method to cluster every half circle. **Table 4.3** shows the parameters of the Euclidean clustering and the clustering results in **Figure 4.6**. Parameters of the Euclidean clustering algorithm greatly affects our proposed 3D tree detection method. They decide whether the method can identify a tree independently. In general, the Euclidean clustering algorithm has three parameters, Euclidean distance, minimum cluster size and maximum cluster size. According to the above average density result of the point cloud map via the downsampling filter, we undertake an experiment to help the clustering algorithm to determine the measurement range of clustering size. As shown in **Table 4.2,** the points number of single layer of the tree trunk and their approximate radius is related. As the trunk radius increases, the number of points for each slice is increased as well.

Table 4.2 The table of the relationship between trunk radius with the number of points in each sliced layer of the trunk.

Radius	**5cm**	**10cm**	**15cm**	**20cm**	**25cm**
No. Points	467±300	789±300	1309±300	1770±300	2384±300

The points are collected from layers of tree trunk sliced into circles, as shown in **Figure 4.6**. It is not accurate, but it can assist the Euclidean clustering algorithm to determine the minimum and maximum cluster size for a better clustering result.

Figure 4.6 The two sliced 2D circles for one layer of an approximately 10cm radius tree trunk with 547 points and 20cm radius tree trunk with 1109.

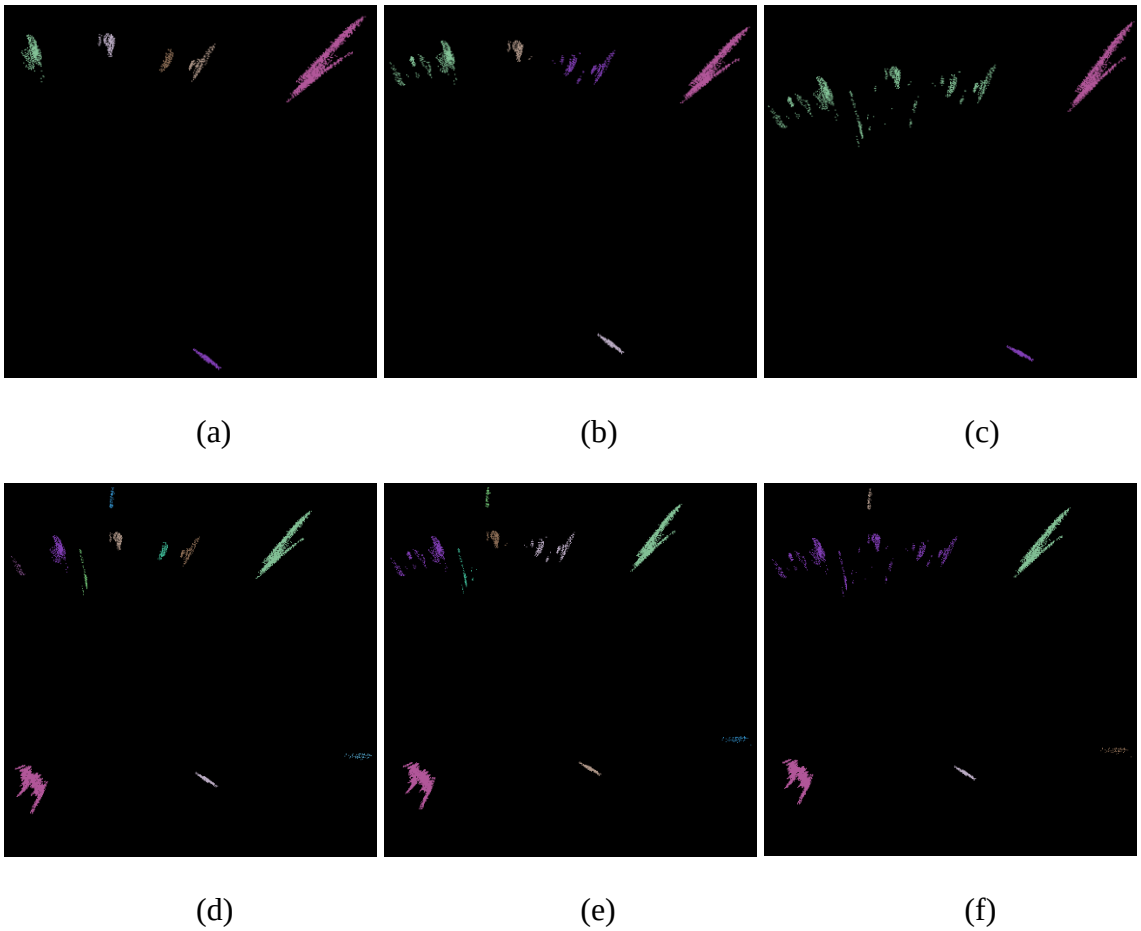
In the research, we need to detect the pine trees which have a 5cm to 20cm radius. Therefore, the minimum and maximum cluster size are set as 50 and 4000. This larger range of values is due to the consideration of noise in the case which can also detect the trunk cross section. As shown in **Table 4.3**, we compare three different Euclidean distance parameters 0.05, 0.1 and 0.2. We also separate the values into two sets of cluster size (200 to 2000) and (100 to 4000). Our purpose is to test the 2D sliced point cloud clustering effect for different Euclidean distances in various cluster size ranges.

Table 4.3 The Euclidean clustering algorithm parameters comparison for the slicing 2D points.

	Euclidean Distance	**Minimum Cluster Size**	**Maximum Cluster Size**
Figure 4.6 (a)	0.05	100	2000
Figure 4.6 (b)	0.1	100	2000
Figure 4.6 (c)	0.2	100	2000

Figure 4.6 (d)	0.05	50	4000
Figure 4.6 (e)	0.1	50	4000
Figure 4.6 (f)	0.2	50	4000

The parameter comparison results of the Euclidean clustering algorithm are shown in following **Figure 4.7** (a - f).



(a) (b) (c)

(d) (e) (f)

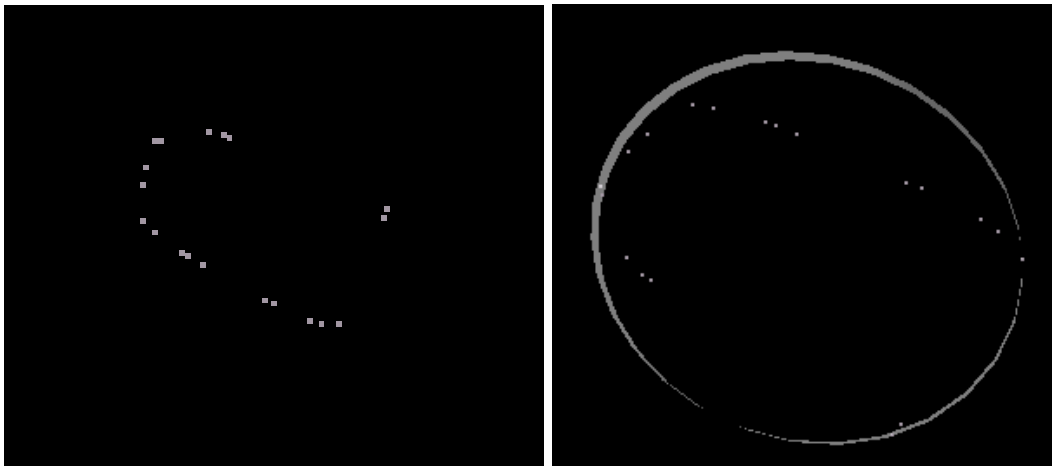
Figure 4.7 Clustering results of the 2D Euclidean clustering algorithm in different Euclidean distance and minimum/maximum cluster size.

(a) Euclidean distance 0.05, min/max cluster size is 100/2000. (b) Euclidean distance 0.1, min/max cluster size is 100/2000. (c) Euclidean distance 0.2, min/max cluster size is 100/2000. (d) Euclidean distance 0.05, min/max cluster size is 50/4000. (e) Euclidean distance 0.1, min/max cluster size is 50/4000. (f) Euclidean distance 0.2, min/max cluster size is 50/4000.

As shown in **Figure 4.7**, the results show that the large Euclidean distance, more than 0.1, mean the clustering algorithm cannot separate the 2D sliced points when this point sets too close. As we can see in (c) and (f), these two images show the 0.2 Euclidean distance

cannot divide the top left area points. The (a) and (b) show that these points can be separated into four or five parts. Therefore, the Euclidean distance is set as 0.05. For the comparison of the cluster size range, the larger range setting can segment more points. Compared with **Figure 4.7** (a) and **Figure 4.7** (d), the image (a) ignores bottom left area points and lower right area points. These two sets of points also represent objects above the ground and should remain. Therefore, we configure the clustering size range from 50 to 4000.

4.1.4.3 2D Circle Fitting



(a) (b)

Figure 4.8 The result of circle fitting. (a) Extracted points from clustered points via convex hull algorithm [Susan et al. 2017]. (b) The visualization of fitted circle via the Levenberg-Marquardt circle fitting algorithm [Chernov in 2010]**.**

After the real clustered point sets are obtained, we use the 2D convex hull algorithm to segment the convex hull points for each set as shown in **Figure 4.8** (a). Moreover, then as mentioned in chapter 3, for each convex hull points set, we use the Levenberg-Marquardt circle fitting algorithm to fit a circle as shown in **Figure 4.8** (b). Finally, through a simple radius filter we obtain a set of circles for the 3D point cloud map as illustrated in **Figure 4.9**.

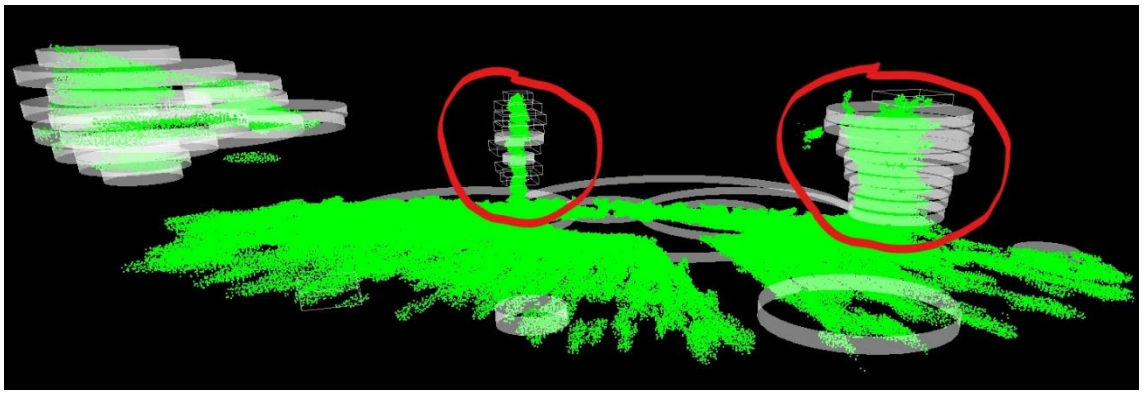


Figure 4.9 The fitted circles in the 3D point cloud map. The two red areas represent the objects. There are two trees in these two spaces.

4.1.4.4 Point Cloud Detection Results

Through the geometric hypothesis of the tree in 3D space, we can determine whether the objects are above the ground or connected to ground. Then we can also define whether the object is a cylinder like object or not. In addition, the limitation of object radius allows the algorithm to retain trees with suitable radius. As shown in **Figure 4.10**, the smaller radius tree is detected, the larger radius tree is ignored.

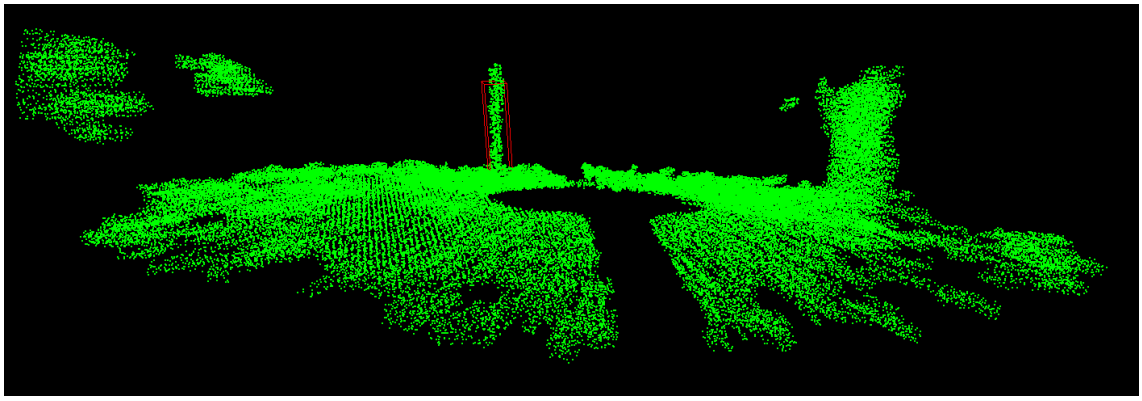


Figure 4.10 The visualization point cloud map of the suitable tree detection. The red cube presents the detected tree position. In the right area, a big tree with a 40cm radius is ignored by the algorithm. In the left area, there is a tree where the trunk of the tree is beyond is the effective distance of the camera. Only the canopy of the tree forms the point cloud.

In the research, we select 20 scenarios with constrained conditions to test the 3D tree detection algorithm based on point cloud. Within the sensor detection range, the limitations of the 20 sets data are:

- No dynamic objects – pedestrians or vehicles.

- Almost flat ground – slope as zero.
- No undergrowth near trees.
- At least one tree within the camera's effective measurement range.

The scenes of the first ten sets of data include only a single suitable tree and a small number of other objects. The scenes of the other ten sets of data include a few suitable trees, a few unsuitable trees and a few other objects within the effective detection range. In the 20 sets of experiment data, there are 32 suitable trees to be detected, ten trees in the ten single tree scenes and 22 trees in the ten multiple tree scenes.

Table 4.4 The result of 3D point cloud tree detection.

	True Positive	**False Positive**
Single Tree Scene	10/10 = 100%	3 errors
Multi-Tree Scene	20/22 = 90.9%	2 errors

Table 4.4 shows the 3D point cloud tree detection rate in the single tree scenes and multi-tree scenes. For the ten single tree scenes, there are ten trees to be detected through the algorithm. However, there are three false positive detection samples in three of these ten single tree experiment scenes. IN one example, as shown in **Figure 4.11**, the right area has a false positive detection sample of a table near a tree.

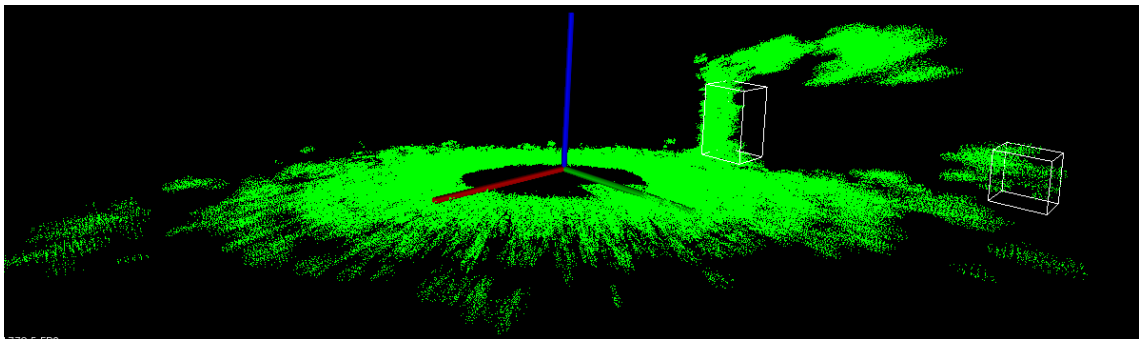


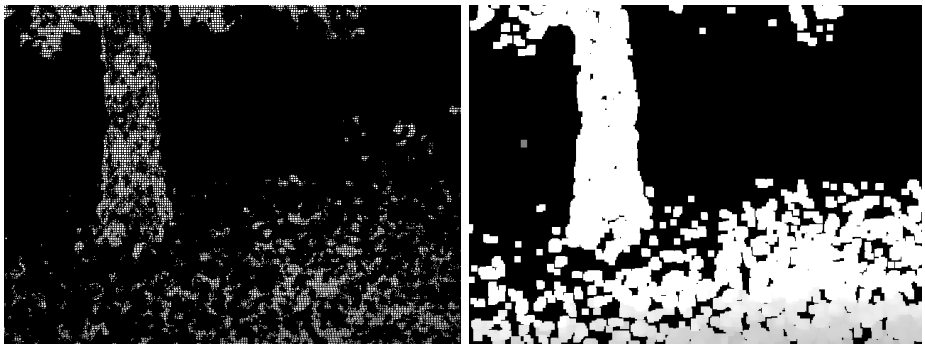
Figure 4.11 The case of false positive tree detection for the 3D tree detection algorithm based on point cloud.

The multiple tree scenes dataset has a 90.9% true positive rate. Because one situation has three trees which are very close, the ORBSLAM2 algorithm generates rough point cloud map through the R200 camera and therefore, the 2D sliced points of the three trees are hard to be clustered by the Euclidean clustering algorithm. In addition, the multiple tree scenes also have two false positive samples. In general, the 3D point cloud tree detection

method can detect trees in both environments, depending on the accuracy of the depth camera and the parameters of the Euclidean clustering algorithm. (In a commercial pine forest, trees are around 5m apart – so there are no trees close to other trees.)

4.1.5 Depth Image Detection

Because the V-SLAM has a cumulative error before the completed loop closure detection finishes, the accuracy of the tree projection from the 3D point cloud into the 2D image is not high enough to be used for our navigation. We use the depth image directly to adjust the relative tree position with a camera. According to the previous three depth image filters, range filter, outlier filter, and ground removal filter, an available depth map can be used to detect trees as shown in **Figure 4.12** (a).



(a) (b)

(c) (d)

Figure 4.12 (a) The registered depth image. (b) The depth image through morphological dilation. (c) The contour of the tree. (d) The final tree detection result.

The purpose of the morphological dilation operation is to ensure that the pixel traversal of the image is not affected by the holes in the depth map. As shown in **Figure 4.12** (b), most of the holes in the image (a) are filled with the dilation operations. Through the detection method mentioned in previous 3.3.2.1, the convex hull of all objects in the camera is detected. Then these convex hulls are filtered through an area and shape filter.

In the resulting example shown in **Figure 4.12** (c), the pink outline represents a contour of the tree. By the result of the previous point cloud detection, the position of the tree in the image is adjusted according to the above depth map detection result. For the depth image tree detection algorithm, the method only considers the areas of the approximate trees from point cloud detection. Through the Kalman filter mentioned in 3.3.2.2, a relatively accurate tree position can be obtained in the image. For example, as shown in **Figure 4.12** (d), the blue rectangle presents the projected point cloud tree detection result, and the green rectangle shows the final tree position generated via the point cloud detection result and depth image detection result.

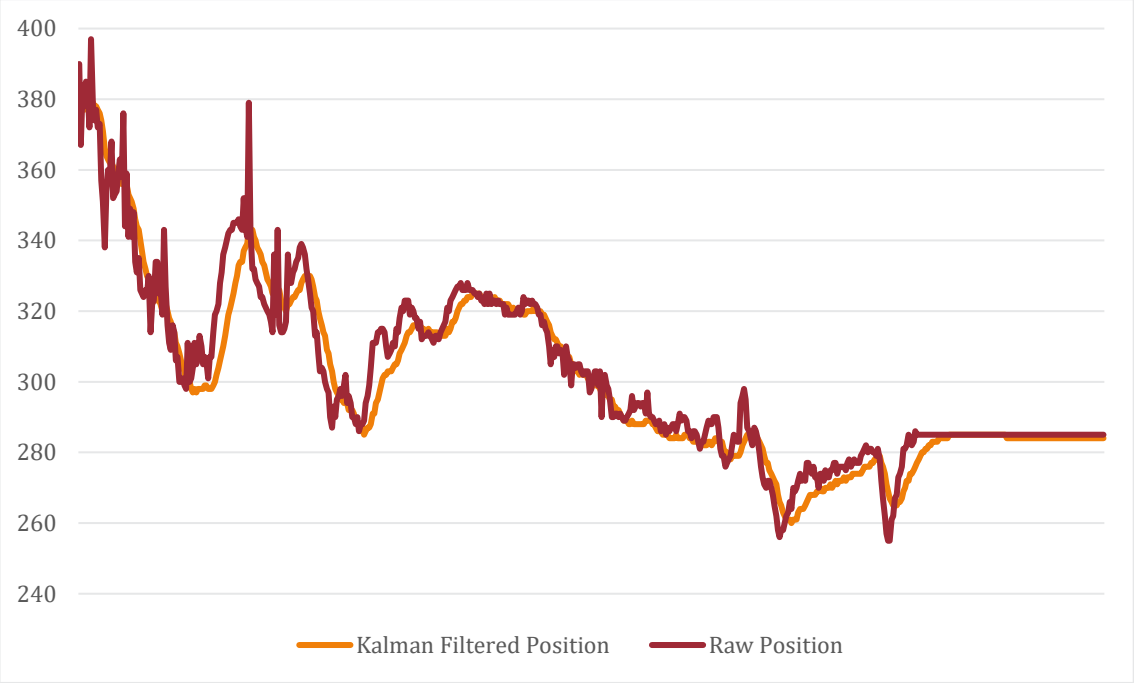


Figure 4.13 The comparison of raw tree position and Kalman filtered data in the x-axis of a 2D image.

Figure 4.13 indicates the process from the toward states to the docked states. The vertical axis represents the position of the detected tree on the x-axis in the screen from only 240 to 400. **Figure 4.13** shows the process from the turn left states to the toward states (i.e. rotating the camera to move the tree from the left area of the image to centre of the image). The vertical axis represents the position of the detected tree on the x-axis in the screen from only 0 to 400.

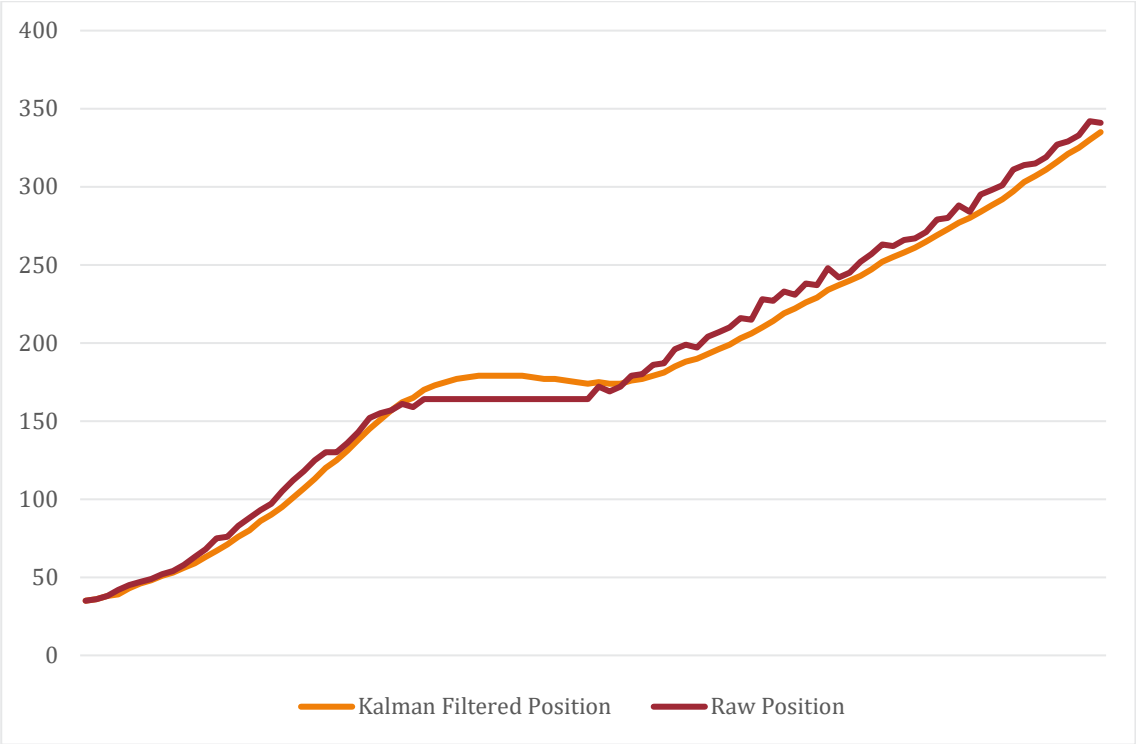


Figure 4.14 The comparison of raw tree position and Kalman filtered data in the x-axis of 2D depth image detection results.

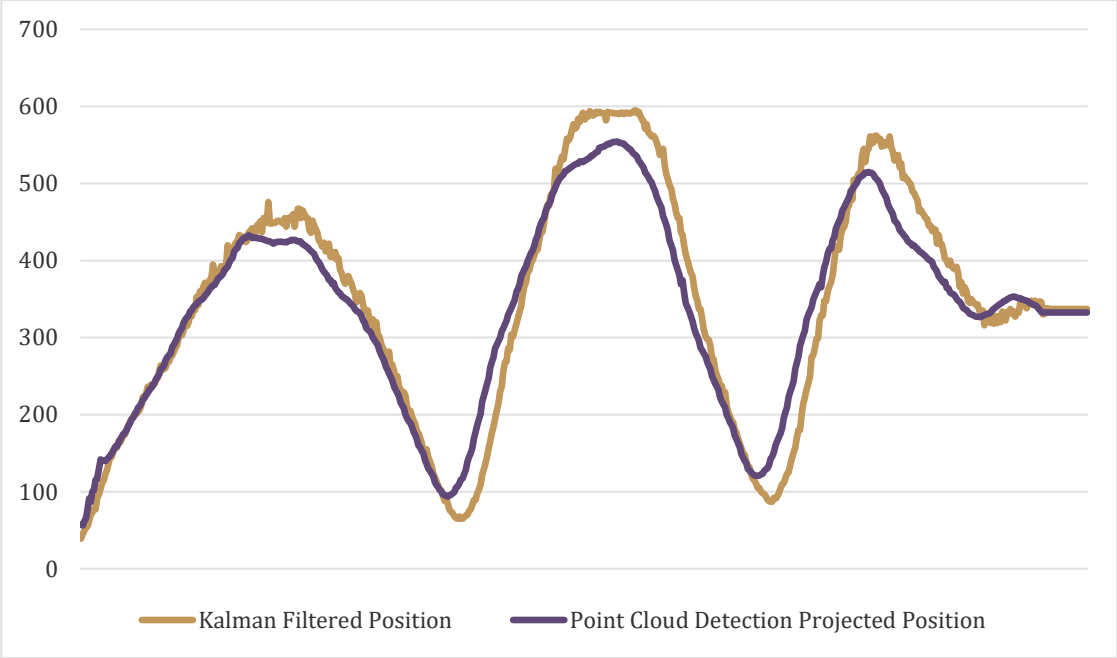


Figure 4.15 The comparison of point cloud tree detection results in purple line and depth image tree detection results in yellow line.

The raw tree position data has significant noise caused by device errors, and camera shake as shown by the red line in **Figure 4.13** and **Figure 4.14**. Through the Kalman filter, the impact of the noise can be reduced to get more smooth detection results for the next

navigation step as shown by the yellow line in the above two figures. Since the depth image tree detection must be based on the point cloud tree detection results, the point cloud detection results have good performance for approximate tree localization. The depth image tree detection can optimize the results to obtain more accurate tree position as shown in **Figure 4.15** and **Figure 4.12** (d).

We test the depth image detection algorithm to run the 20 experiment datasets (10 single tree scenes and ten multiple tree scenes). As shown in **Table 4.5**, the location of each tree is corrected through the depth image detection algorithm from the approximate tree position in the single tree scene experiment. However, in the multiple tree scene experiment, only 15/22 trees are corrected via the algorithm. The complexity of the depth image tree detection is $O(n)$, where n is the number of depth pixels. The time spent traversing the 640×480 depth image through a pointer and reshape method is less than four milliseconds for the experiment laptop based on OpenCV. Therefore, the depth detection method is a real-time detection algorithm. Our proposed slicing point cloud method has a total 93.8% true positive rate for the 20 experiment datasets. Compared with the direct cylinder fitting method, the proposed method increases the point cloud detection correct rate from 81.3% to 93.8%. In addition, this approach also significantly improves the cylinder detection time from an average 11.5 sec (cylinder model fitting method) to 4.1 sec (point cloud slicing method). It can be seen that the proposed 3D point cloud slicing tree detection method has a better performance than the cylinder model fitting method. However, the correction rate of the depth image tree detection method is not good enough to be used for multiple tree environment yet, especially when multiple trees are clustered together. In most of the commercial pine forests in New Zealand, the distance between trees is usually approximately 5 meters. Therefore, this method has a better performance in a commercial pine forest.

In general, the whole tree position estimation algorithm is a hybrid method which estimates the tree position through both point cloud and depth image. According to the different scenes, the algorithm currently can handle the simple single tree environment or trees well-spaced apart.

Table 4.5 The comparison of average time cost and true positive rate of point cloud slicing and cylinder fitting tree detection method through point cloud and depth image.

	Slicing Method Detection Rate	**Cylinder Fitting Rate**	**Depth Image Correction Rate**
Single Tree Scene	10/10 = 100%	9/10 = 90%	10/10 = 100%
Multi-Tree Scene	20/22 = 90.1%	17/22 = 77.3%	15/22=68.2%
Total	93.8%	81.3%	78.1%
Average Time	4.1 sec	11.5 sec	<0.004 sec

4.2 Navigation

4.2.1 Handheld Camera Experiment

The system navigates to the position of the tree in the image and determines the relative position between the camera and the target tree using the above Kalman filter. As shown in **Figure 4.16** (a-c), the red navigation arrow indicates the tree direction for the experimenter who held the camera.

(a) (b) (c)

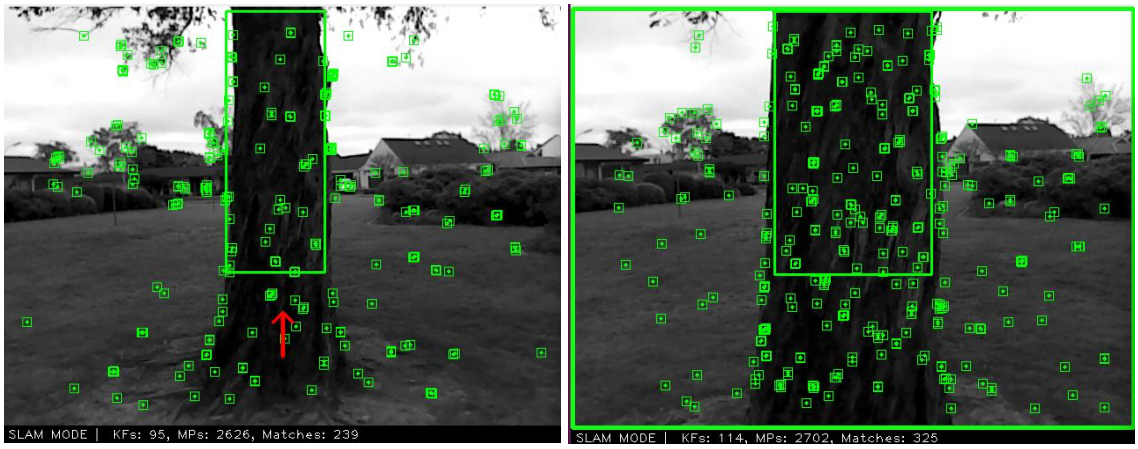
Figure 4.16 The three basic navigation states during the handheld camera experiment, include (a) turn camera left, (b) turn camera right, and (c) go ahead preparation. The red arrow indicates the direction in which the camera needs to be rotated. The green rectangle shows the final detected tree position.

The experimenter moves the camera according to the arrow. Therefore, it finally docks with the target tree at a distance of one meter from the tree as shown in **Figure 4.17**. When the camera is docked with the tree, a green border appears on the screen. In the

whole navigation process, the relative position between the camera and the detected tree can be calculated. The camera is the point in the 3D world coordinate (x_1, y_1, z_1) and the center of the detected tree is a coordinate (x_2, y_2, z_2) as well. Through an equation:

$$\text{distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{22}$$

the distance between the two points can be calculated. When the distance is less than one meter, and the center of the tree is in the middle of the camera field of view, the camera is docked successfully with the tree.



(a) (b)

Figure 4.17 The handheld camera navigation image shows from toward states (a) to docked states (b). The green rectangle shows the detected tree position on the screen, the red arrow shows go ahead at toward states, and the green border around the display shows the docked states.

5 CONCLUSION AND FUTURE WORK

The results of the research show the proposed tree detection program can be used to enable a UAV to dock with a tree using a SLAM based approach and a set of simple navigation commands, using the RGB-D visual SLAM algorithm, ORBSLAM2, to reconstruct a 3D point cloud map. The approximate location of a nearby tree can be detected through the proposed point cloud slicing method. Then according to the approximate tree position and the proposed depth image tree detection method, a more precise tree position method can be measured and visualized in a 2D image. Finally, via the tree position in the 2D image as shown as the green rectangle in **Figure 4.16** and **Figure 4.17**, the navigation ROS node publishes a movement command to control the UAV to dock with the target tree.

ROS is the most popular open source robot meta-operation system. It is a powerful distributed computing framework designed specifically for robots. We use the ROS to support the framework of our proposed method. In addition, we choose an open source flight controller PX4 to work with ROS. Compared with the APM flight controller, PX4 provides ten times the performance of memory and distributed processing methods. We test the proposed method on a UAV simulator, Gazebo 7, which has good compatibility with the flight controller PX4 and ROS.

As part of a larger pine tree pruning project, this thesis presents the first step of a task to locate trees and dock with them. Because the proposed method is based on computer vision technology, we compare several different cameras and decide to use the Intel RealSense R200 depth camera as the visual sensor of the research. In order to get a more accurate measurement of the location and size of the tree, we process the raw depth data from the R200 camera. Through a bilateral filter, a range threshold filter and holes filling algorithm, a relatively reliable depth data can be obtained as the basis for subsequent research.

In the research, we use an existed SLAM framework, ORBSLAM2, to estimate the camera pose and reconstruct a point cloud map. ORBSLAM2 is a very comprehensive open source SLAM algorithm. It has an excellent performance in terms of RGB-D SLAM.

According to the camera pose estimation via ORB-SLAM2, a dense point cloud map is registered for the tree detection.

A point cloud slicing tree detection method is proposed in the research. From our 20 experiments, the proposed method has a 100% correct tree detection rate in the single tree scenes and 90.9% correct tree detection rate in the multiple tree scenes (due to trees being very close together unlike a commercial pine forest where they are well spaced 5m apart). The running time of the proposed point cloud tree detection method is around four seconds (average 4.1 seconds), and the cylinder fitting method is around 10 seconds (average 11.5 seconds) in the 20 experiment datasets.

Compared with prior LiDAR point cloud tree detection methods, the proposed method provided a higher tree detection rate. For example, using LiDAR tree detection methods, [Chen 2007] provided a filtering and forest studies' method (TIFFS) which has a 42%-76% true positive rate. [McGaughey 2012] presented a Forest Service Pacific Northwest Research Station (FUSION) method with a 32%-59% tree detection correct rate. [Ayrey et al. 2017] proposed a layer stacking tree detection method with a 66%-89% correct rate.

In theory, when the camera pose and tree 3D coordination are known, the navigation algorithm provides a 4×4 translation and rotation matrix to enable docking with the tree. However, because visual SLAM algorithm normally has a cumulative error by visual odometry, and the point cloud calculation time is too long, we stop the calculation of the point cloud after obtaining an approximate relative position of the tree with a camera.

Then, we propose a depth image tree detection algorithm to determine a more accurate position of the tree based on the approximate tree location. The iteration time of depth image is fast at around two milliseconds. Therefore, the location of the tree can be processed in real-time.

Finally, we use the Kalman filter to improve the relative position of the tree and the camera. The navigation ROS node publishes movement control commands to enable docking with the tree.

5.1 Future Research

5.1.1 Data Acquisition

It would be useful to use multiple camera viewpoints and merge visual odometry with other sensors such as GPS, IMU, etc.

5.1.2 Point Cloud Mapping

We reduce the density of the point cloud map to enable real-time operation. However, in future research a higher precision more complete map of trees and terrain could be generated through better point cloud registration, point cloud alignment, 2D reflectance map, point cloud objectivization, etc. For the map of the forest, it should include each segmentation of the plant and its surrounding ground situation. Such a high precision complete map could be generated during post-processing.

5.1.3 Complete Navigation System

In the future, this project needs a more complete navigation system to enable UAVs to achieve the pruning task. A complete autonomous navigation system includes three parts, behavior prediction, path planning and obstacle avoidance.

5.1.4 System

In the research, all the tasks are processed on one platform including point cloud map generation, object detection, navigation and all algorithms. We should separate the algorithms, detection, navigation, map generation into four different parts to enable the system to process the surrounding environment in real-time.

ROS is very useful for autonomous navigation, but there are still some problems to be solved such a, reliability, performance, and security. ROS is using a single master node to communicate messages, and there is no monitoring mechanism to recover the failed node. Therefore, the reliability of the system can be improved through building a monitoring mechanism inside the ROS. In addition, when using ROS topic messages to communicate between nodes, the multiple copies of information causes performance degradation. To solve the problem, we can directly use a shared memory instead of the TCP/IP communication mode. Furthermore, ROS does not have authorization and encryption mechanisms, so security is a big threat as well. Thus, we should encrypt the communication message in the future.

6 REFERENCES

[1] "Situation and outlook for New Zealand agriculture and forestry". NZ Ministry of Agriculture and Forestry. 2007. Retrieved June 05, 2017.

[2] Somerville, A. (1991). Pruned Stand Certification. *FRI bulletin*, *167*.

[3] Parker, R., Ashby, L., Tappin, D., & Moore, D. The New Zealand Forest Industry Accident Reporting Scheme (ARS).

[4] Rifkin, J. (2013). The third industrial revolution. *International Study Reference*, *6*, 009.

[5] Azmat, M. M., & Schuhmayer, C. (2015). Self Driving Cars.

[6] Shirai, Y. (2012). *Three-dimensional computer vision*. Springer Science & Business Media.

[7] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In ICRA workshop on open source software (Vol. 3, No. 3.2, p. 5).

[8] O'Kane, J. M. (2014). A gentle introduction to ros.

[9] Garage, W. (2012). Robot operating system (ROS).

[10] Gerkey, B. (2015, December 09). ROS, the Robot Operating System, Is Growing Faster Than Ever, Celebrates 8 Years. Retrieved June 05, 2017.

[11] Greenwald, T. (2016, July 20). Morgan Quigley | Innovators Under 35. Retrieved June 05, 2017

[12] Clearpath Robotics. (2014, January 29). ROS 101: Intro to the Robot Operating System.

[13] Mazzari, Vanessa. "ROS - Robot Operating System." *ROS – Robot Operating System*. Génération Robots, 26 July 2016.

[14] Qi, J., Song, D., Han, J., & Dai, L. (2009). *Design, implement and testing of a rotorcraft UAV system*. INTECH Open Access Publisher.

[15] Open Source Hardware Association. (2014). Open Source Hardware (OSHW) Statement of Principles 1.0.

[16] Mellis, D., Banzi, M., Cuartielles, D., & Igoe, T. (2007, April). Arduino: An open electronic prototyping platform. In *Proc. Chi* (Vol. 2007).

[17] Meier, L., Honegger, D., & Pollefeys, M. (2015, May). PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on* (pp. 6235-6240). IEEE.

[18] Meier, L., Tanskanen, P., Fraundorfer, F., & Pollefeys, M. (2011, May). Pixhawk: A system for autonomous flight using onboard computer vision. In*Robotics and automation (ICRA), 2011 IEEE international conference on* (pp. 2992-2997). IEEE.

[19] Meier, L., Tanskanen, P., Heng, L., Lee, G. H., Fraundorfer, F., & Pollefeys, M. (2012). PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, *33*(1-2), 21-39.

[20] Grabe, V., Riedel, M., Bulthoff, H. H., Giordano, P. R., & Franchi, A. (2013, September). The TeleKyb framework for a modular and extendible ROS-based quadrotor control. In *Mobile Robots (ECMR), 2013 European Conference on* (pp. 19-25). IEEE.

[21] Gültekin, M. K., Erbulan, Ö., Atay, T., Uzun, G., & Ede, E. (2016). AUTOBEE Team Intelligent Ground Vehicle Design Report.

[22] Engelhard, N., Endres, F., Hess, J., Sturm, J., & Burgard, W. (2011, April). Real-time 3D visual SLAM with a hand-held RGB-D camera. In *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Vasteras, Sweden* (Vol. 180, pp. 1-15).

[23] Javier Civera, Oscar G. Grasa, Andrew J. Davison and J. M. M. Montiel: 1-Point RANSAC for EKF Filtering. Application to Real-Time Structure from Motion and Visual Odometry , Journal of Field Robotics, 2010

[24] Engel, J., Schöps, T., & Cremers, D. (2014, September). LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision*(pp. 834-849). Springer International Publishing.

[25] Forster, C., Pizzoli, M., & Scaramuzza, D. (2014, May). SVO: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on* (pp. 15-22). IEEE.

[26] Klein, G., & Murray, D. (2007, November). Parallel tracking and mapping for small AR workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on* (pp. 225-234). IEEE.

[27] Klein, G., & Murray, D. (2008). Improving the agility of keyframe-based SLAM. *Computer Vision–ECCV 2008*, 802-815.

[28] Klein, G., & Murray, D. (2009, October). Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on* (pp. 83-86). IEEE.

[29] Mur-Artal, R., Montiel, J. M. M., & Tardos, J. D. (2015). ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, *31*(5), 1147-1163.

[30] Endres, F., Hess, J., Sturm, J., Cremers, D., & Burgard, W. (2014). 3-D mapping with an RGB-D camera. *IEEE Transactions on Robotics*, *30*(1), 177-187.

[31] Labbe, M., & Michaud, F. (2014, September). Online global loop closure detection for large-scale multi-session graph-based slam. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on* (pp. 2661-2666). IEEE.

[32] Zanuttigh, P., Marin, G., Dal Mutto, C., Dominio, F., Minto, L., & Cortelazzo, G. M. (2016). Operating Principles of Structured Light Depth Cameras. In*Time-of-Flight and Structured Light Depth Cameras* (pp. 43-79). Springer International Publishing.

[33] Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: part I. *IEEE robotics & automation magazine*, *13*(2), 99-110.

[34] Bailey, T., & Durrant-Whyte, H. (2006). Simultaneous localization and mapping (SLAM): Part II. *IEEE Robotics & Automation Magazine*, *13*(3), 108-117.

[35] Gálvez-López, D., & Tardos, J. D. (2012). Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*,*28*(5), 1188-1197.

[36] Labbe, M., & Michaud, F. (2013). Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, *29*(3), 734-745.

[37] Kähler, O., Prisacariu, V. A., & Murray, D. W. (2016, October). Real-Time Large-Scale Dense 3D Reconstruction with Loop Closure. In *European Conference on Computer Vision* (pp. 500-516). Springer International Publishing.

[38] Thrun, S. (1998). Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, *99*(1), 21-71.

[39] Moravec, H. P. (1980). *Obstacle avoidance and navigation in the real world by a seeing robot rover* (No. STAN-CS-80-813). STANFORD UNIV CA DEPT OF COMPUTER SCIENCE.

[40] Matthies, L., & Shafer, S. T. E. V. E. N. A. (1987). Error modeling in stereo navigation. *IEEE Journal on Robotics and Automation*, *3*(3), 239-248.

[41] Alismail, H., & Browning, B. (2009). Exploring visual odometry for mobile robots. 2009).[2010-02-05]. http://www. cs. cmu. edu/afs/cs/user/mjs/ftp/thesis-09/abstracts/alismailEA. pdf.

[42] Nistér, D., Naroditsky, O., & Bergen, J. (2004, June). Visual odometry. InComputer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on (Vol. 1, pp. I-652). IEEE.

[43] Scaramuzza, D., & Fraundorfer, F. (2011). Visual odometry. *IEEE Robotics & Automation Magazine*, *18*(4), 80-92.

[44] Davison, A. J., Reid, I. D., Molton, N. D., & Stasse, O. (2007). MonoSLAM: Real-time single camera SLAM. *IEEE transactions on pattern analysis and machine intelligence*, *29*(6).

[45] Lee, H. S., Kim, K. Y., Kim, T. R., & Park, G. H. (2012). Fast encoding algorithm based on depth of coding-unit for high efficiency video coding.*Optical Engineering*, *51*(6), 067402-1.

[46] Persson, M., Piccini, T., Felsberg, M., & Mester, R. (2015, June). Robust stereo visual odometry from monocular techniques. In *2015 IEEE Intelligent Vehicles Symposium (IV)* (pp. 686-691). IEEE.

[47] Dryanovski, I., Valenti, R. G., & Xiao, J. (2013, May). Fast visual odometry and mapping from RGB-D data. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on* (pp. 2305-2310). IEEE.

[48] DePasqua, L. (2014). *U.S. Patent No. 8,879,359*. Washington, DC: U.S. Patent and Trademark Office.

[49] Zhang, J., & Singh, S. (2015, May). Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 2174-2181). IEEE.

[50] Intel Corporation, "Intel® RealSense™ Camera R200 Product Datasheet", Jun. 2016, Rev. 1.

[51] Fang, Z., & Zhang, Y. (2015). Experimental evaluation of rgb-d visual odometry methods. *International Journal of Advanced Robotic Systems*, *12*.

[52] Geiger, A., Lenz, P., & Urtasun, R. (2012, June). Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on* (pp. 3354-3361). IEEE.

[53] Bailey, T., Nieto, J., Guivant, J., Stevens, M., & Nebot, E. (2006, October). Consistency of the EKF-SLAM algorithm. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on* (pp. 3562-3568). IEEE.

[54] Martinez-Cantin, R., & Castellanos, J. A. (2005, August). Unscented SLAM for large-scale outdoor environments. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on* (pp. 3427-3432). IEEE.

[55] Choi, C., & Christensen, H. I. (2013, November). RGB-D object tracking: A particle filter approach on GPU. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on* (pp. 1084-1091). IEEE.

[56] Grisetti, G., Strasdat, H., Konolige, K., & Burgard, W. (2011). g2o: A general framework for graph optimization. In *IEEE International Conference on Robotics and Automation.*

[57] Koenderink, J. J., & Van Doorn, A. J. (1991). Affine structure from motion.*JOSA A*, *8*(2), 377-385.

[58] Triggs, B., McLauchlan, P. F., Hartley, R. I., & Fitzgibbon, A. W. (1999, September). Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms* (pp. 298-372). Springer Berlin Heidelberg.

[59] Hartley, R., & Zisserman, A. (2005). Multiple view geometry in computer vision. *Robotica*, *23*(2), 271-271.

[60] Nieto, J. I., Guivant, J. E., & Nebot, E. M. (2004, April). The hybrid metric maps (HYMMs): A novel map representation for DenseSLAM. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on* (Vol. 1, pp. 391-396). IEEE.

[61] Engel, J., Sturm, J., & Cremers, D. (2013). Semi-dense visual odometry for a monocular camera. In *Proceedings of the IEEE international conference on computer vision* (pp. 1449-1456).

[62] Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011, November). ORB: An efficient alternative to SIFT or SURF. In Computer Vision (ICCV), 2011 IEEE International Conference on (pp. 2564-2571). IEEE.

[63] Maguya, A. S., Junttila, V., & Kauranne, T. (2013). Adaptive algorithm for large scale dtm interpolation from lidar data for forestry applications in steep forested terrain. *ISPRS journal of photogrammetry and remote sensing*, *85*, 74-83.

[64] Moskal, L. M., & Zheng, G. (2011). Retrieving forest inventory variables with terrestrial laser scanning (TLS) in urban heterogeneous forest. *Remote Sensing*, *4*(1), 1-20.

[65] Sithole, G. (2001). Filtering of laser altimetry data using a slope adaptive filter. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, *34*(3/W4), 203-210.

[66] Mongus, D., & Žalik, B. (2012). Parameter-free ground filtering of LiDAR data for automatic DTM generation. *ISPRS Journal of Photogrammetry and Remote Sensing*, *67*, 1-12.

[67] Yu, Y., Li, J., Guan, H., Wang, C., & Yu, J. (2015). Semiautomated extraction of street light poles from mobile LiDAR point-clouds. *IEEE Transactions on Geoscience and Remote Sensing*, *53*(3), 1374-1386.

[68] Yu, Y., Li, J., Guan, H., & Wang, C. (2015). Automated extraction of urban road facilities using mobile laser scanning data. *IEEE Transactions on Intelligent Transportation Systems*, *16*(4), 2167-2181.

[69] Wang, S. (2016). Urban Roadside Tree Inventory Using a Mobile Laser Scanning System.

[70] Dalponte, M., Bruzzone, L., & Gianelle, D. (2011). A system for the estimation of single-tree stem diameter and volume using multireturn LiDAR data. *IEEE Transactions on Geoscience and Remote Sensing*, *49*(7), 2479-2490.

[71] Yao, W., Krzystek, P., & Heurich, M. (2012). Tree species classification and estimation of stem volume and DBH based on single tree extraction by exploiting airborne full-waveform LiDAR data. *Remote Sensing of Environment*, *123*, 368-380.

[72] Chen, Q. (2007). Airborne lidar data processing and information extraction.*Photogrammetric engineering and remote sensing*, *73*(2), 109.

[73] McGaughey, R. J. (2012). FUSION/LDV: Software for LIDAR data analysis and visualization. Version 3.10. USDA For. Serv. Pacific Northwest Res. Station. Seattle, Wash. Available from http://www. fs. fed. us/eng/rsac/fusion/[accessed June 2012].

[74] Ayrey, E., Fraver, S., Kershaw Jr, J. A., Kenefic, L. S., Hayes, D., Weiskittel, A. R., & Roth, B. E. (2017). Layer Stacking: A Novel Algorithm for Individual Forest Tree Segmentation from LiDAR Point Clouds. *Canadian Journal of Remote Sensing*, 1-13.

[75] Liu, Y., Zhu, S., Jin, B., Feng, S., & Gong, H. (2004, June). Sensory navigation of autonomous cleaning robots. In *Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on* (Vol. 6, pp. 4793-4796). IEEE.

[76] Ram, A., & Santamaria, J. C. (1997). Continuous case-based reasoning.*Artificial Intelligence*, *90*(1), 25-77.

[77] Arleo, A., Smeraldi, F., & Gerstner, W. (2004). Cognitive navigation based on nonuniform Gabor space sampling, unsupervised growing networks, and reinforcement learning. *IEEE Transactions on Neural Networks*, *15*(3), 639-652.

[78] Ge, S. S., & Cui, Y. J. (2002). Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, *13*(3), 207-222.

[79] Jaradat, M. A. K., Garibeh, M. H., & Feilat, E. A. (2009, March). Dynamic motion planning for autonomous mobile robot using fuzzy potential field. In*Mechatronics and its Applications, 2009. ISMA'09. 6th International Symposium on* (pp. 1-6). IEEE.

[80] Faisal, M., Hedjar, R., Al Sulaiman, M., & Al-Mutib, K. (2013). Fuzzy logic navigation and obstacle avoidance by a mobile robot in an unknown dynamic environment. *International Journal of Advanced Robotic Systems*, *10*.

[81] Oh, J. S., Choi, Y. H., Park, J. B., & Zheng, Y. F. (2004). Complete coverage navigation of cleaning robots using triangular-cell-based map. *IEEE Transactions on Industrial Electronics*, *51*(3), 718-726.

[82] Pan, J., Zhang, L., & Manocha, D. (2012). Collision-free and smooth trajectory computation in cluttered environments. *The International Journal of Robotics Research*, 0278364912453186.

[83] Araujo, R. (2006). Prune-able fuzzy ART neural architecture for robot map learning and navigation in dynamic environments. *IEEE Transactions on Neural Networks*, *17*(5), 1235-1249.

[84] Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., & Von Stryk, O. (2012, November). Comprehensive simulation of quadrotor uavs using ros and gazebo. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots* (pp. 400-411). Springer Berlin Heidelberg.

[85] Meng, W., Hu, Y., Lin, J., Lin, F., & Teo, R. (2015, November). ROS+ unity: An efficient high-fidelity 3D multi-UAV navigation and control simulator in GPS-denied environments. In *Industrial Electronics Society, IECON 2015-41st Annual Conference of the IEEE* (pp. 002562-002567). IEEE.

[86] Takaya, K., Asai, T., Kroumov, V., & Smarandache, F. (2016, October). Simulation environment for mobile robots testing using ROS and Gazebo. In*System Theory, Control and Computing (ICSTCC), 2016 20th International Conference on* (pp. 96-101). IEEE.

[87] Brewster, S. (2016). Uber starts self-driving car pickups in Pittsburgh. *TechCrunch, September*.

[88] Reutebuch, S. E., Andersen, H. E., & McGaughey, R. J. (2005). Light detection and ranging (LIDAR): an emerging tool for multiple resource inventory. *Journal of Forestry*, *103*(6), 286-292.

[89] Fujii, K. (2013). Extended Kalman Filter. *Reference Manual*.

[90] Wan, Eric A., and Rudolph Van Der Merwe. "The unscented Kalman filter for nonlinear estimation." *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*. Ieee, 2000.

[91] Sim, R., Elinas, P., Griffin, M., & Little, J. J. (2005, July). Vision-based SLAM using the Rao-Blackwellised particle filter. In *IJCAI Workshop on Reasoning with Uncertainty in Robotics* (Vol. 14, No. 1, pp. 9-16).

[92] Bell, B. M., & Cathey, F. W. (1993). The iterated Kalman filter update as a Gauss-Newton method. *IEEE Transactions on Automatic Control*, *38*(2), 294-297.

[93] Ranganathan, A. (2004). The levenberg-marquardt algorithm. *Tutoral on LM algorithm*, 1-5.

[94] Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., & Burgard, W. (2011, May). g 2 o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on* (pp. 3607-3613). IEEE.

[95] Rusu, R. B., & Cousins, S. (2011, May). 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*(pp. 1-4). IEEE.

[96] Weinmann, M. (2016). Point Cloud Registration. In *Reconstruction and Analysis of 3D Scenes* (pp. 55-110). Springer International Publishing.

[97] Kiranyaz, S., Ince, T., Pulkkinen, J., & Gabbouj, M. (2011). Personalized long-term ECG classification: A systematic approach. *Expert Systems with Applications*, *38*(4), 3220-3226.

[98] Susan Hert and Stefan Schirra. 2D Convex Hulls and Extreme Points. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.10 edition, 2017.

[99] Chernov, N. (2010). Circular and linear regression: Fitting circles and lines by least squares. CRC Press.

[100] Wang, Y. M., Li, Y., & Zheng, J. B. (2010, June). A camera calibration technique based on OpenCV. In *Information Sciences and Interaction Sciences (ICIS), 2010 3rd International Conference on* (pp. 403-406). IEEE.

Websites:

[101] "Journey – Waymo". *Waymo*. N.p., 2017. Web. 9 June 2017. Retrieved June 09, 2017.

[102] Amadeo - Jan 10, 2017 12:20 am UTC, R. (2017, January 09). Google's Waymo invests in LIDAR technology, cuts costs by 90 percent. Retrieved June 09, 2017.

[103] Meier, L. Gagne, D. Willee, H et al. PX4 Development Guide. [website]. Available: https://dev.px4.io/en/. Retrieved June 09, 2017.

[104] Shao, L., Chen, X., Milne, B., & Guo, P. (2014, June). A novel tree trunk recognition approach for forestry harvesting robot. In *Industrial Electronics and Applications (ICIEA), 2014 IEEE 9th Conference on* (pp. 862-866). IEEE.

[105] Kolb, A., Meaclem, C., Chen, X., Parker, R., Gutschmidt, S., & Milne, B. (2015, June). Tree trunk detection system using LiDAR for a semi-autonomous tree felling robot. In *Industrial Electronics and Applications (ICIEA), 2015 IEEE 10th Conference on* (pp. 84-89). IEEE.

7 APPENDICES

APPENDIX A CAMERA CALIBRATION 89

APPENDIX B STRUCTURE FROM MOTION 91

APPENDIX A CAMERA CALIBRATION

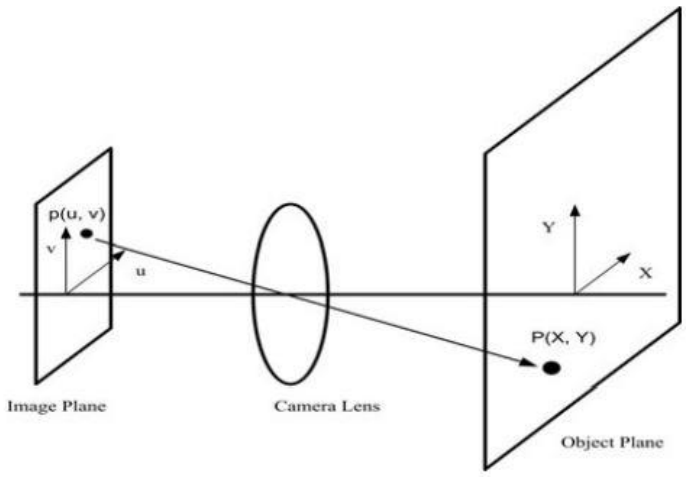


Figure A.1. The concept of camera calibration, provided by [Zhang et al. 2010].

[Zhang et al. 2010] provided a camera calibration method based on OpenCV. The principle of camera calibration is the following step via a checkerboard. In this case, the image stream that captures from the RealSense webcam is converted to a matrix. Thus, one proposed method is used to find the corners of the checkerboard. In this case, the 9×7 checkerboard is used to calibrate the webcam. Then, the found corners of the checkerboard are related between 3D scenes (X, Y, Z) and the 2D image (x, y). Via the relationship of the object points and the image points, point (X, Y, Z) in the 3D world is projected onto the image plane ($\frac{f_x}{Z}, \frac{f_y}{Z}$), and the (u, v) is the principal point. Actually, the projection function is shown as equations (1, 2).

$$\mathrm{x} = \frac{f_x X}{Z} + u \tag{A-1}$$

$$\mathrm{y} = \frac{f_y Y}{Z} + v \tag{A-2}$$

In addition, by using homogeneous coordinates, these equations are enabled to present as matrix format. In the homogeneous coordination, three-dimensional vectors indicate a two-dimensional point. Moreover, a four-dimensional vector indicates a three-dimensional point. The new coordinate is an arbitrary scaling factor (S). In general, a rotation vector and a translation vector are required to transform back to camera coordinates via applying the rigid body transformation in the three-dimensional points. Thus, the projection function could be written as shown following equation 3.

$$S\begin{bmatrix}x\\y\\1\end{bmatrix}=\begin{bmatrix}f_x & 0 & u_0\\0 & f_y & v_0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}r1 & r2 & r3 & t1\\r4 & r5 & r6 & t2\\r7 & r8 & r9 & t3\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix} \quad \text{(A-3)}$$

The rotation value is represented as r1 to r9 and the translation vector represented as t1 to t3. They are the extrinsic parameters of the camera calibration. Finally, a method is used to remove the distortion of the image. Indeed, the method is mapping the input distortion image points to new positions. However, because the transformation is the non-linear transformation, some pixels of the input image is beyond the checkerboard in the image. In the experiment, this excess of the image is ignored.

As shown in **Figure A.2**, the R200 depth camera calibration result is provided includes camera matrix and distortion parameters.

```
[image]

width
640

height
480

[narrow_stereo]

camera matrix
657.343487 0.000000 336.224250
0.000000 661.151311 192.995115
0.000000 0.000000 1.000000

distortion
0.065002 -0.453833 -0.015900 -0.006026 0.000000

rectification
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000

projection
651.557373 0.000000 334.476440 0.000000
0.000000 655.863586 187.673866 0.000000
0.000000 0.000000 1.000000 0.000000
```

Figure A.2. The R200 depth camera calibration results.

APPENDIX B STRUCTURE FROM MOTION

Structure from motion (SfM) is moving a camera in a static environment to get multiple images at a different time. Assuming that these pictures are the projection of the same rigid object, the motion parameters of the camera can be estimated from the image feature correspondences. Bundle adjustment (BA) is the most important step of the SfM. It uses the Levenberg–Marquardt algorithm to minimize the error between the observed image point coordinates and the predicted image point coordinates. As shown in [camera product datasheet] that the core problem of the BA is to minimize the following non-linear re-projection error function (B-1).

$$\min_{a_j, b_i} \sum_{i=1}^{n} \sum_{j=1}^{m} v_{ij} d(Q(a_j, b_i), x_{ij})^2 \quad \text{(B-1)}$$

Assuming that n 3D points in m images, x_{ij} means the i^{th} point projection coordinates on the j^{th} image. v_{ij} means if the x_{ij} has the projection the v_{ij} is 1, else the v_{ij} is 0. Where $Q(a_j, b_i)$ is the prediction value means the b_i projection coordinates based on the camera a_j. The function $d(x, y)$ represents the Euclidean distance between the observed image coordinates and the predicted image coordinates.